


ORIGINAL ARTICLE

Linear work generation of R-MAT graphs

Lorenz Hübschle-Schneider and Peter Sanders* 

Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany (e-mail: huebschle@kit.edu)

*Corresponding author. Email: sanders@kit.edu

Action Editor: Ulrik Brandes

Abstract

R-MAT (for Recursive MATrix) is a simple, widely used model for generating graphs with a power law degree distribution, a small diameter, and community structure. It is particularly attractive for generating very large graphs because edges can be generated independently by an arbitrary number of processors. However, current R-MAT generators need time logarithmic in the number of nodes for generating an edge—constant time for generating one bit at a time for node IDs of the connected nodes. We achieve constant time per edge by precomputing pieces of node IDs of logarithmic length. Using an *alias table* data structure, these pieces can then be sampled in constant time. This simple technique leads to practical improvements by an order of magnitude. This further pushes the limits of attainable graph size and makes generation overhead negligible in most situations.

Keywords: graph generator; parallel processing; large graphs; bit parallelism; sampling

1. Introduction

Graphs are the universal abstraction of relations between objects. With the “big data” revolution, many applications have emerged that have to process huge graphs (e.g., social networks, web graphs, “brain graphs”, gene sequencing data). However, developing algorithms that process large graphs is often limited by the size of the available graphs. Today, the largest real-world networks, such as the Facebook graph, are not available to most researchers. For emerging applications or input sizes anticipated for the future, no real-world inputs are available at all. Hence, scalable graph generators are a useful surrogate. An important model with respect to large graphs is the R-MAT model (Chakrabarti et al., 2004). R-MAT is simple, models power law degree distributions, and also produces graphs with a community structure that is somewhat similar to complex real-world networks. Its simplicity also allows theoretical analysis of its properties, for example, diameter, clustering coefficient, and degree distribution (Mahdian & Xu, 2007; Leskovec et al., 2010). A big advantage of R-MAT is that edges can be generated independently in an embarrassingly parallel way and hence can be used even on large parallel machines. Therefore, R-MAT is used in the Graph 500 benchmark (Murphy et al., 2010), which is the most well-known supercomputer benchmark for graph algorithms and even nonnumerical algorithms in general.

However, recent results on communication-free graph generation (Sanders & Schulz, 2016; Funke et al., 2019; Bläsius et al., 2019) put the role of R-MAT as the most scalable graph generator into question. Other models with similar properties, such as Barabasi–Albert (BA) preferential attachment graphs (Barabasi & Albert, 1999; Sanders & Schulz, 2016) or random hyperbolic graphs (RHGs) (Krioukov et al., 2010; Funke et al., 2019; Bläsius et al., 2019) can now be generated in a massively parallel fashion using only linear work, whereas previous R-MAT generators

| | | | | | | | | | | | | | |
|---|---|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
| a | b | aa | ab | ba | bb | aaa | aab | aba | abb | baa | bab | bba | bbb |
| c | d | ac | ad | bc | bd | aac | aad | abc | abd | bac | bad | bbc | bbd |
| | | ca | cb | da | db | aca | acb | ada | adb | bca | acb | bda | bdb |
| | | cc | cd | dc | dd | acc | acd | adc | add | bcc | bcd | bdc | ddb |
| | | | | | | caa | cab | cba | cbb | daa | dab | dba | dbb |
| | | | | | | cac | cad | cbc | cbd | dac | dad | dbc | dbd |
| | | | | | | cca | ccb | cda | cdb | dca | dcb | dda | ddb |
| | | | | | | ccc | ccd | cdc | cdd | dcc | dcd | ddc | ddd |

Figure 1. Probabilities of adjacency matrix entries for $k = 1, 2,$ and $3.$

need logarithmic time for each edge. Indeed, the highly tuned RHG implementation in Funke et al. (2019) outperforms previous R-MAT generators also in practice. This is surprising since the R-MAT model is very simple while generating RHGs requires comparatively sophisticated geometric data structures and evaluating transcendental functions. Since RHGs are also believed to be “closer” to complex real-world networks, one could question how useful the R-MAT model is besides its now-established role as a standard benchmark. By accelerating R-MAT generation by a logarithmic factor (and by an order of magnitude in practice), we put things back to what one would expect. Generating R-MAT graphs is faster and much simpler than efficient generation of RHGs. It is also similarly fast as generating BA graphs which arguably have a less realistic community structure.

In Section 2, we explain the required concepts and previous algorithms. Section 3 discusses additional related work. Our new algorithm is explained in Section 4. The algorithm precomputes sequences of decisions made by the previous algorithm. These are sampled using a standard technique for weighted sampling to generate graphs with the same distribution as before. Section 5 reports experiments. Our technique can be adapted to a other applications that define random objects recursively. We exemplify this in Section 6 by outlining generalizations to undirected graphs, more general stochastic Kronecker graphs, bipartite graphs, simple graphs, smoothed degree distribution and adaptations to different models of parallel computation. We summarize our findings in Section 7.

2. Preliminaries

The simplest variant of R-MAT defines a directed multi-graph $G = (V, E)$ with $n = 2^k$ nodes and m edges based on four parameters $a, b, c,$ and d with $a + b + c + d = 1.$ Each edge is drawn independently at random using the following recursive process. Consider the $n \times n$ adjacency matrix M of G where $M_{ij} = 1$ if $(i, j) \in E$ and $M_{ij} = 0$ otherwise. M is split into four quadrants. The edge is placed in the upper left quadrant with probability $a,$ in the upper right with probability $b,$ in the lower left with probability $c,$ and in the lower right quadrant with probability $d.$ The process repeats this subdivision k times until a single entry of the adjacency matrix is determined; see also Figure 1.

Our R-MAT generator uses an *alias table* data structure that can be used to sample from any categorical distribution in constant time (Walker, 1977). An alias table can be built in time $\mathcal{O}(u)$ for u elements (categories) (Vose, 1991). Our variant consists of an array A of size $u.$ Each array entry (bucket) represents a total probability weight of $1/u.$ Each bucket b stores two elements e and e' of the set we are sampling. Bucket b also stores a weight w which indicates the part of the probability of element e represented in $b.$ The remaining bucket width $1/u - w$ represents part of the probability of the *alias* element $e'.$ A simple linear time preprocessing algorithm distributes the element probabilities over the buckets such that all buckets are exactly filled and such that the exact probability of each element is represented in some subset of buckets. To sample an element, one generates a random real number $x \in [0, u)$ and considers bucket $b = A[\lfloor x \rfloor].$ If $x - \lfloor x \rfloor < w,$ element e is returned, otherwise the alias e' is returned.¹

3. More related work

There has been a considerable amount of work on generating graphs that is too voluminous to review in detail. We thus concentrate on R-MAT and where it fits into the landscape of graph models. We refer to surveys for further reading (Goldenberg et al., 2010; Drobyshevskiy & Turdakov, 2020; Penschuck et al., 2020) and to Section 6 for some generalizations. In particular, Penschuck et al. (2020) discuss scalability in detail.

R-MAT is one of several popular models that have properties observed in complex real-world networks, such as a small diameter and highly skewed degree distribution. There is a trade-off between how simple the model is, how efficient and simple its generators are, and how well the resulting graphs fit various applications. We see R-MAT as taking a middle-ground with respect to simplicity and realism of the model. On the one hand, there are simple models such as small-world graphs (Watts & Strogatz, 1998) and the BA (preferential attachment) model (Barabasi & Albert, 1999). These models have fast generators (e.g., Sanders & Schulz 2016) but the resulting graphs lack some properties observed in real-world graphs such as a community structure.

On the other hand, there are models that are arguably more realistic than R-MAT but also more complicated and computationally expensive to generate. For example, the RHGs already discussed in the introduction (Krioukov et al., 2010) have a geometric structure that might help to simulate the geographic information in real-world complex networks. The LFR model (Lancichinetti et al., 2008) is an example of a model that allows detailed control of the community structure.

4. Our algorithm

For generating R-MAT graphs, we use an alias table to store precomputed subpaths of the recursive process described in Section 2. For some constant $\alpha < 1$, we precompute $u = n^\alpha$ paths together with their probabilities. By sampling and concatenating the resulting paths, we generate $\Theta(\log n)$ address bits of adjacency array entries in each iteration. The simplest such strategy generates, for a fixed length $\ell < 0.5 \log n$, all $u = 4^\ell$ paths of a length ℓ . These paths can be generated using a simple recursive procedure in time $\mathcal{O}(u)$. The following pseudocode outputs each path representing it as a tuple (i, j, p) where i stands for ℓ bits of the row index of the adjacency matrix, j for ℓ bits of the column index, and p for the probability that this path is generated.

```

Procedure enumItems( $i, j; \{0, 1\}^*, p, \ell$ )           -- Initially ( $\langle \rangle, \langle \rangle, 1, \ell$ )
  if  $\ell = 0$  then output item  $(i, j, p)$ 
  else enumItems( $i \cdot 0, j \cdot 0, ap, \ell - 1$ ); enumItems( $i \cdot 0, j \cdot 1, bp, \ell - 1$ )
        enumItems( $i \cdot 1, j \cdot 0, cp, \ell - 1$ ); enumItems( $i \cdot 1, j \cdot 1, dp, \ell - 1$ )
    
```

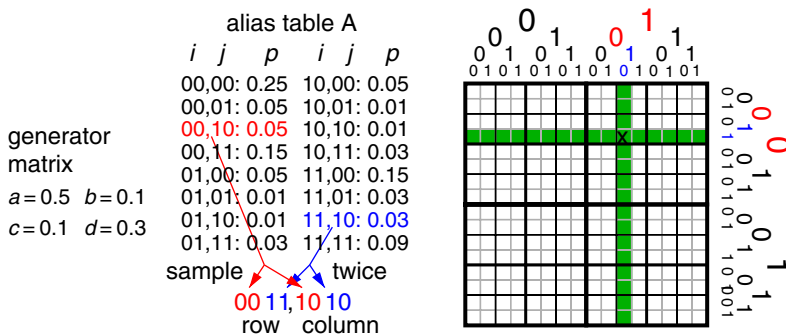


Figure 2. Example for sampling one edge (marked X) from an 8×8 adjacency matrix based on a generator matrix with probabilities $a = 0.5$, $b = d = 0.1$, and $c = 0.3$ using an alias table of size 16 that stores subpaths of length $\ell = 2$.

This pseudocode uses strings of bits for clarity where a real implementation would process machine words in a bit-parallel fashion in constant time. Operator “.” stands for string concatenation.

The resulting sequence of partial paths E is then preprocessed into an alias table A that allows constant time sampling. Full row and column indices of adjacency array entries can then be generated by repeatedly sampling pieces of row and column indices and concatenating them until $k = \log n$ bits are available. After at most $\lceil k/\ell \rceil \in \mathcal{O}(1)$ iterations, a new edge will be completed. Figure 2 gives an example. By reusing leftover bits from generating previous edges, we can indeed see that generating B edges will take $\lceil kB/\ell \rceil$ sampling operations. The following pseudocode implements this approach to generate a batch of B edges:

```

i := j := {}          -- fragments of row and column index bit strings, initially empty
e := 1
while e ≤ B do       -- Generate a batch of B edges.
    (i', j') := A.sample() -- get more bits using alias table A
    i := i · i'; j := j · j' -- append them to known bits
    if |i| ≥ k then    -- enough bits for a new edge
        output edge (i[0..k - 1], j[0..k - 1])
        i := i[k..]; j := j[k..] -- reuse remaining bits for
        e := e + 1      -- next edge

```

Overall, we get the following result:

Theorem 1. *The above algorithm computes an R-MAT graph with $m \in \Omega(n)$ edges in time $\mathcal{O}(m)$.*

Proof. The correctness of the algorithm is evident from the above description. Since the number of items is $4^\ell < 4^{0.5 \log n} = 2^{\log n} = n \in \mathcal{O}(m)$, generating items and preprocessing them into an alias table (Walker, 1977) can be done in time $\mathcal{O}(m)$ (Vose, 1991). As argued above, generating each edge takes constant time. \square

In the technical report (Hübschle-Schneider & Sanders, 2019a), we also describe a generalization that generates bit strings of nonuniform length. This might be faster for highly skewed generator matrices.

5. Experiments

We perform experiments on a machine with two Intel Xeon E5-2683 v4 processors with 16 processing cores each. With hyperthreading (two hardware-supported threads on each core) this means we can use up to 64 threads. Our implementation is written in C++ and compiled with the GNU C++ compiler g++ in version 8.2.0. The source code is available at <https://github.com/lorenzhs/wrs/tree/rmat>. We compare with the R-MAT generators used in the Graph 500 benchmark reference implementation and in the NetworKit toolkit for large-scale network analysis (Staudt et al., 2016). We removed the code for making the graph undirected and for scrambling node IDs in order to concentrate on the core task of R-MAT: edge generation. The modified code is also available under the above link.

Figure 3 shows the throughput using 64 threads as a function of the available alias table size. For $n = 2^{30}$ nodes, we generate 10^{11} edges overall, assigning blocks of 2^{16} edges at a time to the threads. We see two peaks in throughput which correspond to fully using L2 cache and L3 cache, respectively. Overall, the best throughput is up to 881 million edges per second.² This is 14.2 times faster than the Graph 500 generator and 7.9 times faster than NetworKit.

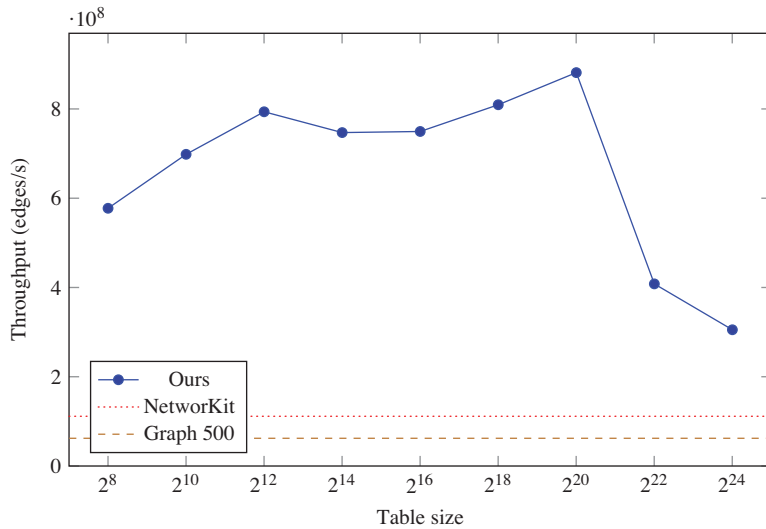


Figure 3. Throughput of R-MAT generator as a function of table size, using 64 threads.

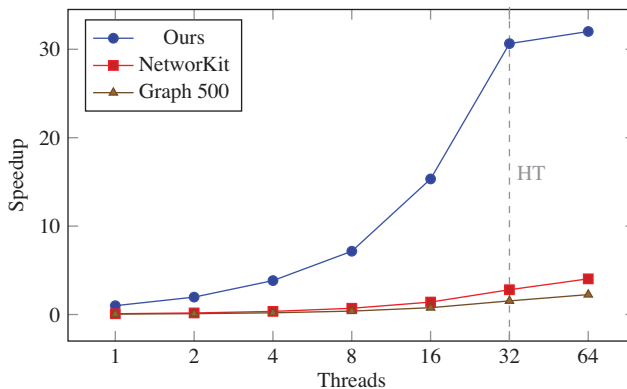


Figure 4. Speedup relative to the fastest sequential algorithm.

Figure 4 gives the speedup over our algorithm on a single thread. We achieve speedup 30.6 on 32 threads and speedup 32 on 64 threads. Hyperthreading is of little help because the memory bandwidth to L3 cache is becoming a limiting factor in our code. In contrast, the Graph 500 and NetworKit generators benefit significantly from hyperthreading since the number of arithmetic operations far dominate the memory accesses.

Figure 5 shows the degree distribution of the generated graphs for all three generators. It is clearly visible that the output graphs of all generators have identical degree distributions. All three generators show the plateaus in the degree distribution that are characteristic of R-MAT graphs. This corroborates our claim that our method generates graphs according to the R-MAT model.

To demonstrate the extreme scale of the graphs that can be generated using our method, we additionally executed it on 256 nodes of SuperMUC-NG, the 9-th fastest supercomputer according to the November 2019 Top 500 list.³ Each node has two Intel Xeon Platinum 8174 processors with 24 cores (48 hardware threads) and 96 GiB of RAM. We ran both our generator and the Graph 500 generator for 20 minutes each, for $n = 2^{44}$ nodes. Our generator produced $2^{47.995}$ edges, 12.4 times more than the $2^{44.361}$ edges produced by the Graph 500 generator. Thus, generating an

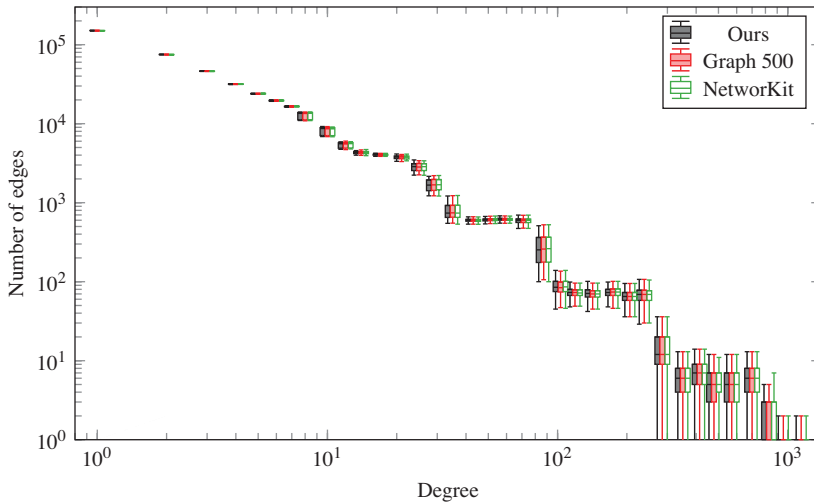


Figure 5. Degree distribution of the generated graphs. Aggregated over bins of width $\lfloor \log_2(d) - 2 \rfloor$ for degree d , 500 runs with $m = 10^7$ edges and $n = 2^{20}$ nodes each. Box plot with quantiles 0.25, 0.5, and 0.75, with whiskers to $1.5 \times$ inter-quantile range, with small horizontal offsets to *Ours* and *NetworKit* for readability.

R-MAT graph as used in the Graph 500 BFS benchmark for scale 44 ($n = 2^{44}$, $m = 2^{48}$) takes just 20 minutes on 256 nodes of SuperMUC-NG. This graph is four times larger than the largest previously used graph in the official Graph 500 benchmark so far, at scale 42 ($n = 2^{42}$, $m = 2^{46}$)⁴.

The speed of our generators compares favorably to generators for other models. For example, the sequential Erdős–Renyi generator of Funke et al. (2019) is only around 10% faster than our sequential generator, but for a much simpler model (the special case $a = b = c = d$). On the other hand, the streaming hyperbolic graph generator sRHG of Funke et al. (2019), a more complicated model, takes around 5 times more time per edge.

6. Generalizations

Several generalizations are quite easy. We can generate an *undirected graph* by mirroring edges in the lower left triangular matrix to the upper right triangular matrix, that is, when an edge (i, j) with $i > j$ is generated, then it is interpreted as the edge (j, i) . Thus, only an upper triangular adjacency matrix M is generated, where entry $M_{i,j}$ with $i < j$ stands for the undirected edge $\{i, j\}$. The Graph 500 benchmark *scrambles* row and column IDs in order to “hide” the structure of the graph. An edge (i, j) generated using the recursion from Section 2 is scrambled by outputting it as $(\pi(i), \pi(j))$ where π is a simple pseudorandom permutation that can be computed in constant time. More general graphs can be generated using a $k \times k$ *generator matrix* rather than a 2×2 matrix for the recursion.

Another view on R-MAT graphs is that its recursive definition specifies a probability p_{ij} for an edge to connect nodes i and j . This view, for a $k \times k$ generator matrix, is also known as *stochastic Kronecker graphs* (Leskovec & Faloutsos, 2007). The R-MAT edge generation process then amounts to sampling from this distribution with replacement, that is, when sampling multiple edges, several of them can connect the same nodes. By *removing duplicate edges*, we change the process to sampling without replacement. A standard way for doing this is to keep a hash table of generated edges and to check that table before outputting an edge. Alternatively, we can use the communication-free distributed algorithm outlined below.

Besides sampling with and without replacement, we can also use Bernoulli sampling to sample each possible edge (i, j) with probability p_{ij} . However, the best known algorithm for this process (Moreno et al., 2018) takes logarithmic time per generated edge.

Chakrabarti et al. (2004) propose to *smooth* the degree distribution of the graph by perturbing the parameters a – d in each step of the process. Implementing this exact approach in constant time per edge seems difficult. However, we can perturb the probabilities for the entries of the alias table A . Furthermore, we can use several such perturbed tables to achieve a similar effect.

One can also preprocess the alias table in parallel (Hübschle-Schneider & Sanders, 2019b). More practically relevant is that parallel graph algorithms work best if each processor (PE) is responsible for all edges incident to a given number of nodes. Sorting the edges accordingly after they were generated would destroy the communication-free property of our generator. We can get a *communication-free* generator for such *partitioned graphs* using an approach similar to the one used for Erdős–Rényi graphs in Funke et al. (2019). The adjacency matrix is split into a number of quadratic *tiles*. We use a divide-and-conquer algorithm that determines the number of edges generated in each of these tiles. Each PE only recurses on those subproblems that intersect the rows of the adjacency matrix assigned to it. The base case is a single tile of the adjacency matrix where we use the algorithm from Section 4 to generate the prescribed number of entries. Note that this makes duplicate elimination a local operation so that this feature can also be implemented in a communication-free way.

7. Conclusions

We give a simple and practical algorithm for generating R-MAT graphs with constant time per edge in an embarrassingly parallel way. This makes this widely used family of graphs even more easily usable for experiments with huge graphs. We improve sequential throughput by an order of magnitude over the basic algorithm by precomputing chains consisting of many random decisions and storing them in an appropriate discrete distribution. The same approach is likely to work in other settings, too.

Acknowledgments. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de).

Conflict of interest. The authors have nothing to disclose.

Notes

1 The alias tables described in the literature are slightly different since they return indices rather than the elements themselves. We deviate from this convention since we need to generate data that is not from an integer range and translating from an index to the actually needed data incurs additional memory access overhead.

2 On an AMD Epyc 7551P with 32 cores (64 hardware threads), we get similar results—slightly better overall performance than on the Intel machine, with the best performance when the L3 cache on each chiplet is fully used.

3 <https://www.top500.org/list/2019/11/>, retrieved on April 17, 2020.

4 https://graph500.org/?page_id=781, retrieved on April 17, 2020.

References

- Barabasi, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509–512.
- Bläsius, T., Friedrich, T., Katzmann, M., Meyer, U., Penschuck, M., & Weyand, C. (2019). Efficiently generating geometric inhomogeneous and hyperbolic random graphs. In *27th European symposium on algorithms (ESA)* (pp. 21:1–21:14).
- Chakrabarti, D., Zhan, Y., & Faloutsos, C. (2004). R-MAT: A recursive model for graph mining. In *SIAM conference on data mining (SDM)* (pp. 442–446). SIAM.

- Drobyshevskiy, M., & Turdakov, D. (2020). Random graph modeling. *ACM Computing Surveys*, 52(6), 131:1–131:36.
- Funke, D., Lamm, S., Meyer, U., Penschuck, M., Sanders, P., Schulz, C., ..., von Looz, M. (2019). Communication-free massively distributed graph generation. *Journal of Parallel and Distributed Computing*, 131, 200–217.
- Goldenberg, A., Zheng, A. X., Fienberg, S. E., & Airolidi, E. M. (2010). A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2), 129–233.
- Hübschle-Schneider, L., & Sanders, P. (2019a). Linear work generation of R-MAT graphs. arXiv/CoRR, [arXiv:1905.03525](https://arxiv.org/abs/1905.03525).
- Hübschle-Schneider, L., & Sanders, P. (2019b). Parallel weighted random sampling. In *27th European symposium on algorithms (ESA)* (pp. 59:1–59:24). LIPIcs.
- Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., & Boguná, M. (2010). Hyperbolic geometry of complex networks. *Physical Review E*, 82(3), 036106-1-036106-18.
- Lancichinetti, A., Fortunato, S., & Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4), 046110.
- Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., & Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11, 985–1042.
- Leskovec, J., & Faloutsos, C. (2007). Scalable modeling of real graphs using Kronecker multiplication. In *24th international conference on machine learning (ICML)* (pp. 497–504). ACM.
- Mahdian, M., & Xu, Y. (2007). Stochastic kronecker graphs. In *5th international workshop on algorithms and models for the web-graph (WAW)* (pp. 179–186). Springer.
- Moreno, S., Pfeiffer, J. J., & Neville, J. (2018). Scalable and exact sampling method for probabilistic generative graph models. *Data Mining and Knowledge Discovery*, 32(6), 1561–1596.
- Murphy, R. C., Wheeler, K. B., Barrett, B. W., & Ang, J. A. (2010). Introducing the Graph 500. *Cray user's group*.
- Penschuck, M., Brandes, U., Hamann, M., Lamm, S., Meyer, U., Safro, I., ..., Schulz, C. (2020). Recent advances in scalable network generation. arXiv/CoRR, [arXiv:2003.00736](https://arxiv.org/abs/2003.00736).
- Sanders, P., & Schulz, C. (2016). Scalable generation of scale-free graphs. *Information Processing Letters*, 116(7), 489–491.
- Staudt, C. L., Sazonovs, A., & Meyerhenke, H. (2016). NetworKit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4), 508–530.
- Vose, M. D. (1991). A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering (TSE)*, 17(9), 972–975.
- Walker, A. J. (1977). An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3), 253–256.
- Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684), 440.