The Algorithmics Column

BY

Gerhard J Woeginger

RWTH Aachen
Lehrstuhl für Informatik 1
D-52056 Aachen, Germany
woeginger@cs.rwth-aachen.de

Large-scale Graph Generation and Big Data: An Overview on Recent Results

Ulrich Meyer[*] Manuel Penschuck[*]

Abstract

Artificially generated input graphs play an important role in algorithm engineering for systematic testing and tuning. In big data settings, however, not only processing huge graphs but also the efficient generation of appropriate test instances itself becomes challenging. In this context we survey a number of recent results for large-scale graph generation obtained within the DFG priority programme SPP 1736 (Algorithms for Big Data).

1 Introduction

Across disciplines, graphs are a universal method to model entities and their relationships: search engines, social networking sites, e-commerce platforms and other businesses regularly keep their real world data in the form of graphs. Similarly, the internet, neurological systems such as the human brain, reactions of pharmaceutical substances or subatomic particles, and biological networks are commonly viewed that way. As the methods of information acquisition and storage continuously improve, an ever increasing amount of data needs to be handled and processed.

Here we focus on the aspects of experimental algorithmics related to theses big data challenges. Algorithms processing massive graphs often use approximations, partition the input into small local chunks, or rely on hierarchical decompositions. Thus, empirical results on their global performance and quality obtained with small instances typically do not generalise to much larger inputs.

Unfortunately, suited massive real world datasets are scarce, difficult to manage and distribute, do not scale, may exhibit noise and uncontrollable properties, often lack ground-truth, and can pose privacy or legal issues. Synthetic instances based on random models help to alleviate these problems. As such models are parametrised and randomly sample instances from a large families, they additionally allow for reproducible, systematic and statistically significant experiments.

[*]Goethe University Frankfurt, {umeyer, mpenschuck}@ae.cs.uni-frankfurt.de

The specific types of random graph models used in a given engineering process highly depend on the application and the experiments carried out. For instance, many real-world graphs (e.g., communication or social networks) exhibit basic features such as a small diameter, a skewed degree distribution, and a non-vanishing local cluster coefficient. Further, some models not only provide a graph, but also additional information which can be used to benchmark an algorithm's performance (e.g., LFR and CKB benchmarks for community detection [9, 19, 20]). The application-specific requirements gave rise to a large body of random graph models which often compromise between mathematical tractability, realism, and sampling costs [2, 31].

We survey a number of new sampling techniques developed by projects of the DFG priority programme SPP 1736 on *Algorithms for Big Data* (see also Section 5) and collaborations between them. While most target large graph instances exceeding main memory, there are also continued efforts for internal memory solutions such as the NetworKit library [29] (SPP), as they support the prototyping of big data algorithms. In order to handle the memory footprint of massive graphs, the following models of computation have been used:

- The *External-Memory model* by Aggarwal and Vitter [1] is a commonly accepted framework to reason about the effects of memory hierarchies. While it is typically used in a scenario with main memory and disks (EM), the design patterns promoting data locality often also lead to more cache efficient algorithms. The model features a two-level memory hierarchy with fast internal memory (IM) which may hold up to M data items, and a slow disk of unbounded size. The measure of an algorithm's performance is the number of I/Os required. Each I/O moves a block of B consecutive items between memory levels. Reading or writing n contiguous items from or to disk requires $\text{scan}(n) = \Theta(n/B)$ I/Os. Sorting n consecutive items triggers $\text{sort}(n) = \Theta((n/B) \cdot \log_{M/B}(n/B))$ I/Os. For all realistic values of n, B and M, $\text{scan}(n) < \text{sort}(n) \ll n$. Sorting complexity constitutes a lower bound for most intuitively non-trivial EM tasks [1, 23].

- Streaming models approach inputs exceeding internal memory by imposing a view even more restricted than the external memory framework [11]. While there are a number of formulations, they have in common that the input may only be observed in a scanning fashion (i.e. read from the beginning to end). For some, up to $O(\log n)$ passes are acceptable where n is the number of tokens in the input stream. Additionally, the memory consumption of the algorithm should be highly sub-linear.

- In a distributed setting, there are a number of processing units with a small local memory. Accessing information not stored there is costly as it requires

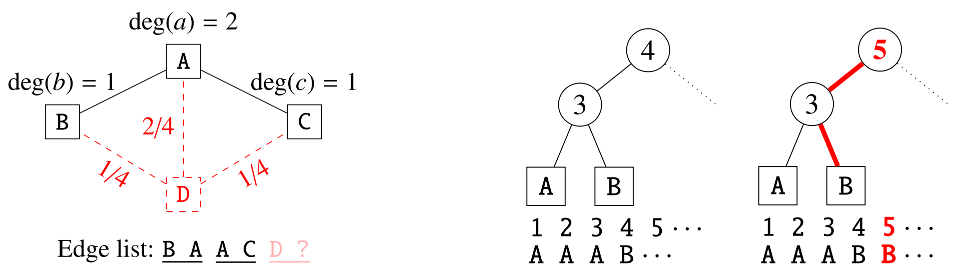

Figure 1: Preferential Attachment. **Left**: Vertex D is added to a seed graph (nodes {A, B, C}) in a Barabási and Albert step. Due to its higher degree, A is twice as likely become a neighbour. Observe that this sampling corresponds to uniformly drawing a node from the edge list. **Right**: Execution of *look-up-and-connect-to* for weight rank four. The query emits an edge to node B and increases its weight.

communication. Hence, a common design goal is to minimise the number of messages exchanged (similarly to I/Os in the external memory model). An extreme case are *communication-agnostic* algorithms where each processing unit is aware of its rank, the total number of units, and some input configuration but cannot exchange any further information.

From now on, let n be the number of nodes and m the number of edges in a graph.

2 Preferential attachment

Preferential attachment is a property found in a number of iterative random processes where nodes with higher degrees are more likely to gain additional edges, which then further increases their probability to be sampled again. In [22] (SPP) we introduced two graph generators in the External-Memory Model (TFP-BA and MP-BA) requiring $\mathcal{O}(\text{sort}(m))$ I/Os to generate m edges with such a stochastic process. Both are demonstrated for (but not limited to) the popular and minimalistic preferential attachment family by Barabási and Albert [4]. The model iteratively growths a random graph from an arbitrary seed graph. In each round a new vertex is introduced and connected to a fixed number of existing nodes randomly drawn with a probability proportional to their current individual degrees.

Batagelj and SPP member Brandes [5] observed that such a distribution can be sampled by choosing a node uniformly at random from the edge list produced so far since each node u appears exactly deg(u) time in it (cf. Fig. 1). Since this method incurs unstructured accesses, it triggers $\Omega(m)$ I/Os with high probability for edge lists exceeding the main memory size M by a constant factor. It is hence infeasible in the external memory setting.

TFP-BA extends this algorithmic idea. It interprets each index of the generated edge list as a check point at which information can be retrieved. The algorithm then relies on Time Forward Processing[1] to send two types of messages to them. Firstly, TFP-BA randomly draws all positions from which it will later sample even before the values to be stored there have actually been computed. It then sends *read messages* to these positions informing them for which later indices their values will be required.

Similarly, *write messages* carry the value that needs to be written to a certain index. Initially, each edge in the seed graph is translated into two write instructions (one for each incident node). In the main phase, the algorithm generates the edge stream. Each entry receives exactly one write command containing its value, and a (possibly empty) set of read tokens which can then be answered accordingly by emitting new write tokens.

While the preferential attachment problem seems inherently sequential, our second algorithm MP-BA combines external-memory techniques with distributed or shared-memory parallelism. Rather than performing operations on the edge list, it explicitly stores the degree distribution in a binary search tree, reducing the memory footprint from $\Theta(m)$ to $\Theta(n)$. Leaves correspond to nodes and store the nodes' degrees as weights. Each inner node tracks the sum of weights in its left sub-tree, thereby effectively maintaining a prefix sum over weight intervals. This enables efficient sampling by drawing a uniform variate from the total weight interval and finding the associated leaf.

The tree can be implemented in external or distributed memory, and allows for a lazy *look-up-and-connect-to* operation (similarly to a Buffer Tree [3] but with improved constants for this tailor-made operation). MP-BA's data structure is suited for all random sampling methods (including the majority of preferential attachment models) which perform linear updates to the underlying probability distribution after each trail. This is due to the fact that we can compute the linear change to a sum of weights without knowledge of individual constituents (cf. Fig. 1). Therefore, we may delay all operations at any inner node in a queue.

In an experimental evaluation on a heterogeneous CPU/GPU system, MP-BA is 17.6 times faster than the sequential internal memory scheme of [5] for instances fitting in main memory, and scales well in the EM setting. We also show an increase in efficiency of nearly one order of magnitude compared to the parallel and distributed approach by [6].

Subsequently, inspired by recent work of Shun et al. [28] on parallelism in randomised incremental algorithms, SPP members Sanders and Schulz published

[1]Time Forward Processing is a generic technique in external-memory algorithms which relies on some topological order of a problem's (implicit) dependency graph. Based on the rank in this order, a checkpoint can send messages forward to points with higher ranks using a priority queue [21]. In TFP-BA ranks are identified with edge list indices.

a communication-agnostic, hashing-based preferential attachment approach [26] that exhibits even more parallelism, albeit in a more restricted setting.

3 LFR benchmark and beyond

The LFR model [19] is a de facto standard benchmark for community detection algorithms. It constructs a graph with a planted cluster structure and additionally emits the memberships of each node as ground-truth to assess the quality of a clustering algorithm.

In order to do so, the degrees of all nodes and sizes of communities are randomly drawn from two parameterisable powerlaw distributions. Subsequently, each node randomly joins a community it fits in. In case of overlapping communities, multiple memberships per vertex are possible. The benchmark then provisions a fraction (controlled by the global mixing parameter) of each node's edges for neighbours outside its own communities, while the remainder stays within its clusters. This is achieved, by first independently generating and subsequently merging the inter-community graph and intra-communities networks. Here it has to be ensured that the inter-community graph does not contain intra-community edges. Also, multi-edges may need to be handled in the overlapping case.

In [15] (SPP collaboration), we describe and analyse a pipeline consisting of four I/O-efficient algorithms which in combination match the LFR model. During the generation of m edges it triggers $\tilde{O}(\text{sort}(m))$ I/Os with high probability. In an experimental evaluation, our implementation outperforms the original generator by at least two orders of magnitude for large instances still fitting into internal memory, and scales to graphs exceeding main memory.

The most demanding stage in our pipeline is the sampling of a graph from a prescribed degree sequence. In accordance with the original LFR generator, we employ the *fixed degree sequence model* (FDSM) which works in two steps: Firstly, a highly biased simple graph is materialised using a deterministic algorithm by Havel and Hakimi [13, 16]. It is then randomised as described in the next section.

3.1 Graph randomisation

The graph randomisation takes a simple graph G which represents the family of all simple graphs with the same degrees as in G. The task is to output a uniform sample. This problem is not limited to LFR: for instance, it also arises in network analysis where randomisation is used to estimate the significance of an observation relative to the family of graphs with the same degree sequence [27] (SPP).

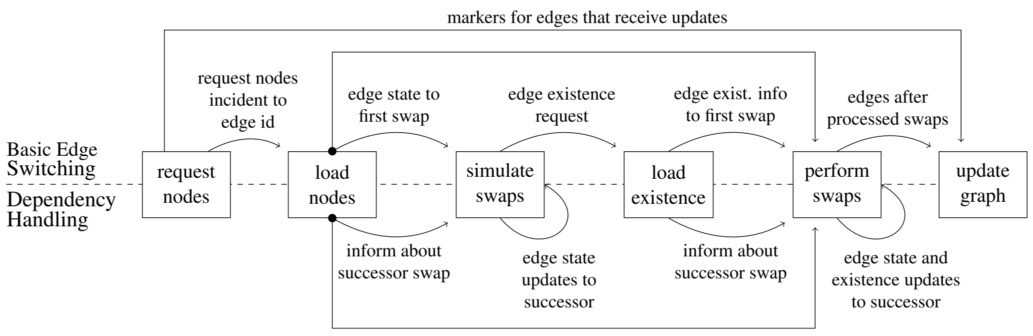
markers for edges that receive updates
request nodes incident to edge id
edge state to first swap
edge existence request
edge exist. info to first swap
edges after processed swaps
Basic Edge Switching
Dependency Handling
request nodes
load nodes
simulate swaps
load existence
perform swaps
update graph
inform about successor swap
edge state updates to successor
inform about successor swap
edge state and existence updates to successor

Figure 2: EM-ES: Stages executed for each batch of swaps.

In [15], we rely on the *edge switching markov chain* to match the original LFR scheme. It applies a list of $c \cdot m$ swaps[2] to a graph with m edges where c is a constant (typically in the range of 10 to 100). Each swap mixes the incident nodes of two random edges. To ensure that the graph remains simple, a swap has to be skipped if it would produce a multi-edge or self-loop. Therefore, the following steps are necessary for each switch:

1. Randomly draw two edges and gather their incident nodes.
2. Mix the nodes and skip if a self-loop arises.
3. Check whether a new edge already exists in the graph and skip if this is the case to prevent multi-edges.
4. Update the graph, i.e. delete the old edges and insert the new ones.

Observe that all steps but the second one access the edge list in a unstructured manner. Our I/O-efficient algorithm EM-ES solves this issue by executing batches of $\Theta(m)$ swaps. Each batch roughly follows the steps outlined above but answers look-ups for all swaps combined (cf. Fig. 2).

Additional care has to be taken to deal with dependencies and conflicts arising if two swaps operate on the same edge. Before we actually perform the swaps, we determine in a simulation phase which configurations are possible and then load and organize all information potentially required. We show (and confirm experimentally) that the incurred overhead is small and that each batch triggers $\mathcal{O}(\text{sort}(m))$ I/Os with high probability.

In ongoing research we propose to integrate the configuration model [24] into our LFR pipeline. This graph randomisation is fast but does not produce simple graphs. However, multi-edges and self-loops can be removed with targeted edge

[2]The number of swaps is a common choice based on empirical results. To the best of our knowledge, there is no rigorous theoretical results for general graphs near the empirically found bounds [8].

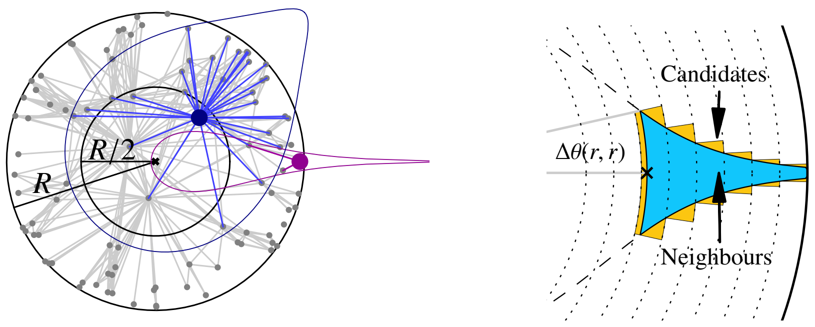

Figure 3: **Left**: A random hyperbolic graph with n=150 nodes. The area enclosed by the two *hyperbolic circle* (lobes) corresponds to all points in distance at most R around its highlighted centre. **Right**: Band model introduced by [34] (not to scale). Neighbours can be found in the inner partial lobe while the step-wise overestimation corresponds to the area considered during candidate selection.

swaps (at least for the required graph classes). In our empirical study [14] (SPP collaboration), we compare the classic FDSM pipeline to one containing these building blocks and find a substantial speed-up due to the extensions.

3.2 Graph replication

Staudt et al. introduce the graph replication method ReCoN [30] (SPP collaboration). It requires a graph and its community structure (which may be supplied using a clustering algorithm) as input. Inspired by LFR, it generates and randomises inter- and intra-community networks with parameters matching the original graph. Outputs larger than the input can be obtained by copying individual cluster parameters to new communities (and adapting the degree sequence accordingly). While these copies locally share a similar structure, they are in general not identical due randomisation steps and unequal memberships. In an experimental evaluation, ReCoN is shown to capture the input graph structure and to output similar networks orders of magnitude larger. While not studied in their work, all steps are compatible with the I/O-efficient techniques outlined above.

4 Hyperbolic random graphs

Random hyperbolic graphs [17] are a popular model for social networks as they *naturally* capture their features. Each node is associated with a point randomly placed inside a hyperbolic circle of radius $R = \Theta(\log n)$. As shown in Fig. 3, most points lie near the boundary as the radial probability density function grows exponentially towards the outside.

In the simplest formulation (threshold model, cf. [12]) two nodes are adjacent iff their distance is at most R. Also in some extensions (e.g., [17]) the probability that any two points are adjacent is a continuous function depending on their distance. For both, the geometric embedding intuitively yields a skewed degree distribution. Due to the hyperbolic distances, the few points near the origin cover a much larger probability mass than a point at the boundary and hence have a higher degree. Additionally, there is a non-vanishing clustering coefficient as two neighbours of some node are likely to be relatively close to each other as well.

A naïve generator for hyperbolic graphs enumerates all $\binom{n}{2}$ pairwise distances and emits an edge for each pair of points close enough. It incurs a sequential runtime of $\Theta(n^2)$ and is hence prohibitively expensive for large n. All sub-quadratic algorithms we are aware of rather identify a set of neighbour candidates based on some geometric overestimation. In a second step, edges are generated by computing only the distances between a node and its candidates.

Here, we present contributions of three SPP projects. We are currently seeking to combine and extend these results in a joint effort.

4.1 Fast generators in internal memory

Looz et al. [33] (SPP) project all points from the hyperbolic plane into a Poincaré disk which allows neighbourhood queries based on euclidean circles. Candidates are then selected using a polar quad-tree. The authors bound the generator's runtime to $O((n^{3/2}+m)\log n)$ with high probability. This approach is compatible with the threshold model and all probabilistic extensions with monotonic decreasing edge probability functions. In [32] (SPP) the techniques are extended to general probabilistic geometric sampling with suggested applications in facility location problems and sensor networks.

Later, Looz et al. introduce a new candidate selection for the threshold model which significantly reduces the runtime by removing the angular separation of the quad-tree [34] (SPP). As sketched in Fig. 3, their generator decomposes the hyperbolic plane into $\Theta(\log n)$ bands whose heights decrease geometrically towards higher radii. In a preprocessing step, the points are randomly scattered over the plane and inserted into the appropriate bands based on their radii. The points are then sorted by their angular coordinates independently for each band.

In the main phase, all points search their neighbours in their own bands and all above them. Every time, the minimal and maximal polar angles in which neighbours can be found in a particular band are computed. Based on these, the first and last candidate in each band can be identified using binary searches. The authors empirically establish a runtime of $O(n \log n + m)$.

4.2 Streaming

In [25] (SPP), we prove a runtime of $O(n \log\log(n) + m)$ for a generator very similar to the band-wise sampling of Looz et al.. It uses bands of constant height and improves data locality by avoiding binary searches. This algorithm is optimal for all graphs with an average degree[3] of $\Omega(\log\log n)$. We also propose method to enumerate all edges in a streaming fashion with a memory footprint of $O([n^{1-\alpha}d^{\alpha} + \log n] \log n)$ where d is the average degree and $\alpha > 1/2$ a parameter controlling the powerlaw degree distribution's exponent $2\alpha{+}1$ with typically $\alpha \ll 2$. For realistic networks with an average degree of $o(n/\log^{1/\alpha}(n))$, this significant reduction is especially useful in the context of co-processors (e.g., GPUs or Xeon Phi) with a relatively small internal memory. On commodity hardware, the implementation produces $3.7 \cdot 10^8$ edges per second for graphs with $10^6 \leq m \leq 10^{12}$ and $\alpha{=}1$, utilising less than 600 MB of RAM.

4.3 Distributed

Lamm [18] (SPP) introduced communication-agnostic distributed sampling schemes for a number of models such as Erdös-Rényi graphs and random geometric networks with euclidean and hyperbolic embeddings. In the sequential case, the hyperbolic generator has an expected linear runtime complexity and can be interpreted as an extension of [7]. It partitions the hyperbolic space into discrete bands of buckets. Their number is chosen such that each cell is expected to contain k points, where $k \approx 4$ is a tuning parameter. The generator allows all processing units to compute the points within any bucket independently, eliminating the need for communication. Lamm bounds the generation time of the distributed grid structure to $O(P \log n + n/P)$, where P is the number of processors, and empirically finds a time-complexity of $O(\frac{n+m}{P} + P \log n)$ for the parallel algorithm.

5 Priority Programme SPP 1736

The results presented above have been obtained within the Priority programme [10] SPP 1736 (Algorithms for Big Data), which has been established by the German Research Foundation (DFG) in 2013 for a total funding period of six years basically split into two funding periods of three years each.

[3]Hyperbolic random graphs not meeting this assumption are typically not connected and have degenerate powerlaw degree distribution with a significant number of singletons. So a different model may be better suited.

Starting in 2014, fifteen research projects plus a coordination project (totalling about 20 full PhD student positions) have been funded during the first period. Most of these research projects will also receive funding within the second period which just starts now. Additionally, a few more projects with own funding have been associated in order to benefit from collaboration and joint events (workshops, PhD meetings, summer schools etc.) organised by the SPP.

Most of the funded projects concentrate on big data algorithms that are not machine learning and frequently tackle more than one of the following areas: (1) technological challenges, (2) fundamental algorithmic techniques, and (3) applications. In particular, the projects investigate algorithmic exploitation of parallelism (multicores, GPUs, parallel and distributed systems, etc.), handling external and outsourced memory as well as memory-hierarchies (clouds, distributed storage systems, hard-disks, flash-memory, etc.), dealing with large scale dynamic data updates, processing compressed data, approximation and online processing under resource constraints. Concrete applications include 3D+T terabyte image analysis, large-scale text processing and semantic search, drug design, and genome assembly.

In spite of covering different areas, the vast majority of projects actually works with graphs and is highly interested in efficient ways to generate artificial graph data. This has already led to a number of ongoing collaborations with joint publications and software packages some of which we reviewed above.

More detailed information about SPP 1736 can be found on the webpage of the programme: `https://www.big-data-spp.de`.

References

[1] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM, 31(9)*, pages 1116–1127, 1988.

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[3] Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. `doi:10.1007/s00453-003-1021-x`.

[4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct 1999. `doi:10.1126/science.286.5439.509`.

[5] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005. `doi:10.1103/PhysRevE.71.036113`.

[6] Hasanuzzaman Bhuiyan, Jiangzhuo Chen, Maleq Khan, and Madhav V. Marathe. Fast parallel algorithms for edge-switching to achieve a target visit rate in heterogeneous graphs. In *ICPP 2014*, pages 60–69, 2014. `doi:10.1109/ICPP.2014.15`.

[7] Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *CoRR*, abs/1511.00576, 2015. URL: `http://arxiv.org/abs/1511.00576`.

[8] Corrie Jacobien Carstens. *Topology of complex networks: models and analysis*. PhD thesis, RMIT University, 2016.

[9] Kyrylo Chykhradze, Anton Korshunov, Nazar Buzun, Roman Pastukhov, Nikolay Kuzyurin, Denis Turdakov, and Hangkyu Kim. *CompleNet 2014*, chapter Distributed Generation of Billion-node Social Graphs with Overlapping Community Structure, pages 199–208. Springer International Publishing, Cham, 2014. `doi:10.1007/978-3-319-05401-8_19`.

[10] DFG, German Research Foundation. Priority Programmes. URL: `http://www.dfg.de/en/research_funding/programmes/coordinated_programmes/priority_programmes/index.html`.

[11] Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 2016. `doi:10.1007/978-3-540-28608-0`.

[12] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering - (extended abstract). In *ICALP 2012*, pages 573–585, 2012. `doi:10.1007/978-3-642-31585-5_51`.

[13] Seifollah L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. i. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962. `doi:10.1137/0110037`.

[14] Michael Hamann, Ulrich Meyer, Manuel Penschuck, Hung Tran, and Dorothea Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. *CoRR*, abs/1604.08738, 2017. URL: `http://arxiv.org/abs/1604.08738`.

[15] Michael Hamann, Ulrich Meyer, Manuel Penschuck, and Dorothea Wagner. I/O-efficient generation of massive graphs following the LFR benchmark. In *ALENEX 2017*, pages 58–72, 2017. `doi:10.1137/1.9781611974768.5`.

[16] Václav Havel. Poznámka o existenci konečných grafů. *Časopis pro pěstování matematiky*, 080(4):477–480, 1955. URL: `http://eudml.org/doc/19050`.

[17] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, Sep 2010. `doi:10.1103/PhysRevE.82.036106`.

[18] Sebastian Lamm. Communication efficient algorithms for generating massive networks. Master's thesis, Karlsruhe Institute of Technology, 2017. `doi:10.5445/IR/1000068617`.

[19] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80:016118, Jul 2009. `doi:10.1103/PhysRevE.80.016118`.

[20] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, Oct 2008. doi:10.1103/PhysRevE.78.046110.

[21] Anil Maheshwari and Norbert Zeh. A survey of techniques for designing I/O-efficient algorithms. In *Algorithms for Memory Hierarchies*, pages 36–61, 2003. doi:10.1007/3-540-36574-5_3.

[22] Ulrich Meyer and Manuel Penschuck. Generating massive scale-free networks under resource constraints. In *ALENEX 2016*, pages 39–52, 2016. doi:10.1137/1.9781611974317.4.

[23] Ulrich Meyer, Peter Sanders, and Jop F. Sibeyn, editors. *Algorithms for Memory Hierarchies, Advanced Lectures [Dagstuhl Research Seminar, March 10-14, 2002]*, volume 2625 of *Lecture Notes in Computer Science*. Springer, 2003.

[24] Michael Molloy and Bruce A. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6(2/3):161–180, 1995. doi:10.1002/rsa.3240060204.

[25] Manuel Penschuck. Generating practical random hyperbolic graphs in near-linear time and with sub-linear memory. In *16th International Symposium on Experimental Algorithms, SEA 2017*, 2017.

[26] Peter Sanders and Christian Schulz. Scalable generation of scale-free graphs. *Inf. Process. Lett.*, 116(7):489–491, 2016. doi:10.1016/j.ipl.2016.02.004.

[27] Wolfgang E. Schlauch, Emőke Ágnes Horvát, and Katharina A. Zweig. Different flavors of randomness: comparing random graph models with fixed degree sequences. *Social Network Analysis and Mining*, 5(1):1–14, 2015. doi:10.1007/s13278-015-0267-z.

[28] Julian Shun, Yan Gu, Guy Blelloch, Jeremy T. Fineman, and Phillip B. Gibbons. Sequential random permutation, list contraction and tree contraction are highly parallel. In *Proc. 26th Symp. on Discrete Algorithms, SODA*, pages 431–448, 2015. doi:10.1137/1.9781611973730.30.

[29] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508âĂŞ530, 2016. doi:10.1017/nws.2016.20.

[30] Christian L. Staudt, Michael Hamann, Ilya Safro, Alexander Gutfraind, and Henning Meyerhenke. *Generating Scaled Replicas of Real-World Complex Networks*, pages 17–28. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-50901-3_2.

[31] Remco Van Der Hofstad. Random graphs and complex networks. *Available on http://www.win.tue.nl/rhofstad/NotesRGCN.pdf*, page 11, 2009.

[32] Moritz von Looz and Henning Meyerhenke. Querying probabilistic neighborhoods in spatial data sets efficiently. In *IWOCA 2016*, pages 449–460, 2016. doi:10.1007/978-3-319-44543-4_35.

[33] Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. In *ISAAC 2015*, pages 467–478, 2015. doi:10.1007/978-3-662-48971-0_40.

[34] Moritz von Looz, Mustafa Safa Özdayi, Sören Laue, and Henning Meyerhenke. Generating massive complex networks with hyperbolic geometry faster in practice. In *HPEC 2016*, pages 1–6, 2016. doi:10.1109/HPEC.2016.7761644.