Generating Scaled Replicas of Real-World Complex Networks

Christian L. Staudt, Michael Hamann, Ilya Safro, Alexander Gutfraind and Henning Meyerhenke

Abstract Research on generative models plays a central role in the emerging field of network science, studying how statistical patterns found in real networks can be generated by formal rules. During the last two decades, a variety of models has been proposed with an ultimate goal of achieving comprehensive realism for the generated networks. In this study, we (a) introduce a new generator, termed ReCoN; (b) explore how models can be fitted to an original network to produce a structurally similar replica, and (c) aim for producing much larger networks than the original exemplar. In a comparative experimental study, we find ReCoN often superior to many other state-of-the-art network generation methods. Our design yields a scalable and effective tool for replicating a given network while preserving important properties at both micro- and macroscopic scales and (optionally) scaling the replica by orders of magnitude in size. We recommend ReCoN as a general practical method for creating realistic test data for the engineering of computational methods on networks, verification, and simulation studies. We provide scalable open-source implementations of most studied methods, including ReCoN.

1 Introduction

Context. When engineering algorithms, the ability to create good synthetic test data sets is valuable to estimate effectiveness and scalability of the proposed methods. A shortage of real data for this purpose can for example arise if they are proprietary,

Christian L. Staudt (e-mail: christian.staudt@kit.edu)✉ · Michael Hamann (e-mail: michael.hamann@kit.edu)✉ · Henning Meyerhenke (e-mail: meyerhenke@kit.edu)✉
Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Ilya Safro (e-mail: isafro@clemson.edu)✉
School of Computing, Clemson University, Clemson, SC, USA

Alexander Gutfraind (e-mail: agutfraind.research@gmail.com)✉
Loyola University Medical Center, Maywood, IL, USA/Uptake, Inc., Chicago, IL, USA;

H. Cherifi et al. (eds.), *Complex Networks & Their Applications V*,
Studies in Computational Intelligence 693,
DOI 10.1007/978-3-319-50901-3_2

sensitive, or unavailable in different scales. In the context of developing network analysis algorithms, realistic synthetic graphs allow us to produce experimental results that are representative for what can be observed for real data. Among the main use cases are *obfuscation* (replacing restricted real data with similar synthetic data), *compression* (storing only a generator and its parameters instead of large graphs), as well as *extrapolation and sampling* (generating data at larger or smaller scales).

Problem definition. We envision two usage scenarios: Given an original (or real) network $O = (V, E)$ ($n_o = |V|$, and $m_o = |E|$) that cannot be freely shared, we would like to be able to create a synthetic network R (with n_r nodes) that matches the original in essential structural properties, so that computational results obtained from processing this network are representative for what the original network would yield. We refer to R as a *replica*. We assume that whoever creates the replica has access to O and can pass it to a *model fitting* algorithm which uses it to parametrize a generative model.

More importantly, in addition to producing *scale-1 replicas* (where $n_r = n_o$), in the second scenario we want to use the generative model for *extrapolation*: We want to parametrize it so that it produces a *scaled replica* R^x that has $n_r = x \cdot n_o$ nodes, where x is called the *scaling factor*. The structural properties of R^x should be such that they resemble a later growth stage of the original (also see Sec. 2). This should enable users of the replica to extrapolate the behavior of their methods when the network data is significantly scaled.

Finally, with respect to performance, we would like the generator algorithm and implementation as well as the fitting scheme to be efficient enough to produce large data sets (on the order of several millions of nodes and edges) quickly in practice.

State of the art. Many generative models for complex networks exist. We point the interested reader to a survey [12] for a more comprehensive overview. A widely used model intended for model fitting uses exponential random graph models (ERGM), cf. e. g. [25]. Unfortunately, ERGM are so expensive that graphs with tens of thousands of nodes are already considered big for these models [3].

Other generative models admit fast generators and are thus in our focus. Among those models are RMAT [6], BTER [16], and Hyperbolic Unit Disk Graphs (HUDG) [17]. Initially, they can fit only few properties of the original network by design, though. A previous fitting scheme by Leskovec et al. [20] for RMAT graphs is quite time-consuming already for medium-sized networks [28, 29].

Editing models create a synthetic network by editing the original network. The MUSKETEER generator [14] implements a multiscale editing model and is effective for obfuscation purposes. However, its current implementation [13] is not fast enough to generate sufficiently scaled replicas of large graphs.

Outline and contribution. In this paper we develop and evaluate a sufficiently fast generator that focuses on creating realistic *scaled* replicas of complex networks.

We point out in Section 2 which criteria we consider important for calling a (scaled) replica realistic. In particular we conceptualize realism in two ways: (i) matching an original graph in a set of important structural properties, and (ii) matching the running time behavior of various graph algorithms.

Our new generator ReCoN, short for *Replication of Complex Networks* and described in Section 3, uses and extends ideas of LFR, a generator used for benchmarking community detection algorithms. Using the original degrees and a found community structure we are able to capture a much-more detailed signature of the network than a parametrization of the LFR generator. In Section 4 we discuss the generative models that we use for comparison (among them RMAT, HUDG, and BTER) and develop model fitting schemes for them.

Our comparative experimental study in Section 5 indicates that ReCoN performs overall quite well and usually better than other generators in terms of realism. We can also conclude that the ReCoN implementation is fast, as it is capable of creating realistic scaled replicas on the scale of 10^8 edges in minutes. The ReCoN code is publicly available in the open-source network analysis package NetworKit [31].

2 Realistic Replicas

We consider a generative model realistic if there is high structural similarity between the synthetic graphs produced and relevant real-world networks. It is neither our goal nor generally desirable to obtain an exact correspondence between original and replica. First, this would exclude the use case of obfuscation. Secondly, obtaining an isomorphic graph is rarely required for generalizable experiments. Note that we consider a single "realism score" for each model inappropriately reductionist. Rather, we quantify diverse aspects of realism in our experimental evaluation and leave it to the reader to decide about their relative importance.

For 1-scale replicas (with the same size as the original), we measure the similarity in terms of a set of commonly used metrics: Sparsity (number of edges vs number of nodes); degree distribution (more precisely its Gini coefficient); maximum degree as a proxy for the connectedness of hub nodes; average local clustering coefficient to measure the local presence of triangles; diameter to monitor the small-world effect; number of connected components and number of communities as additional non-local features. These metrics cover both local and global properties and are deemed important characteristics of networks [23].

How can we extend the notion above regarding realism to *scaled* replicas of a network? To answer this question, let us look at the scaling behavior of a set of 100 Facebook social networks [32]. These networks were collected at an early stage of the Facebook online social networking service in which networks were still separated by universities. Fig. 1 plots basic structural measures of these Facebook networks against the number of nodes n, as well as a regression line and confidence intervals (shaded area) to emphasize the trend. While linear regression may not always seem completely appropriate for these data, the general trend is still captured.

We can observe from Fig. 1 a growth of the number of edges m that is linear in n, an increase in the skew of the node degree distribution as measured by the Gini coefficient, a growing maximum node degree, a slightly falling average local clustering coefficient, a nearly constant small diameter of the largest connected component, and a slightly growing number of connected components (which can be explained

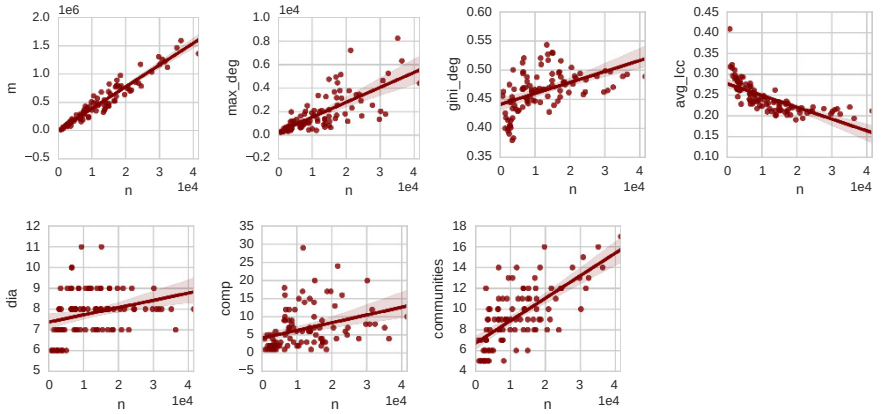

Fig. 1: Scaling behavior of 100 Facebook networks; from left to right and top to bottom: number of edges, maximum degree, Gini coefficient of degree distribution, average local clustering coefficient, diameter, number of components, number of communities found by PLM

by some small connected components that exist in addition to a giant component). We detect communities using PLM (Parallel Louvain Method), a modularity-based community detection heuristic [30], and report the number of communities minus the number of these small connected components. It can be observed that the number of non-trivial communities grows slightly.

While we do not propose that these scaling laws are universal, the trends represented here are commonly observed [4, 5, 27]. Thus, we use them to define desired scaling properties for the remainder of the study as follows: m grows linearly with n; the diameter does not change significantly, preserving the "small world property"; the shape of the degree distribution remains skewed; the maximum node degree increases; the number of connected components may grow; the number of communities increases slightly.

Recall that one use case for our generator is testing of graph and network analysis algorithms. Since the running time is an essential feature in such tests, we also consider a realistic replication of running times important. To this end, we select a set of graph algorithms that (i) compute important features of networks and are thus frequently used in network analysis tasks and that (ii) cover a variety of patterns of computation and data access, each of which may interact differently with the graph structure. The set consists of algorithms for connected components (essentially breadth-first search), PageRank (via power iteration), betweenness approximation (according to Geisberger et al. [11]), community detection (PLM, [30]), core decomposition (according to [9]), triangle counting (according to [15]), and spanning forest (essentially Kruskal's algorithm without edge weights).

3 The Generation Algorithm ReCoN

We introduce ReCoN, a generator for replicating and scaling complex networks. Its input is a graph and a community structure on it. For fitting a given graph without given community structure, we use PLM [30] in order to detect a community structure first. The basic idea of ReCoN is to randomize the edges inside communities and the edges between communities while keeping the node degrees. This happens separately such that each community keeps as many edges as it had before. For scaling a graph, we first create as many disjoint copies of the graph as desired and then apply the aforementioned steps. During the randomization of the edges between the communities the copies usually become connected with each other.

The idea of randomizing graphs inside and between communities is inspired by the LFR generator, a benchmark graph generator for community detection algorithms [19]. There the basic building blocks are also a random subgraph per community and a global graph. However, in the LFR generator the degrees and communities are not given but generated using a power law degree distribution and a power law community size distribution with nodes assigned to communities at random, while ReCoN uses the given graph as input for them.

For randomizing graphs while preserving the degree sequence we use random edge switches where two edges $\{u,v\}$, $\{y,z\}$ chosen uniformly at random are changed into $\{u,z\}$, $\{y,v\}$ if the resulting graph is still simple, i. e. does not contain any duplicate edges or self-loops. Similar to the edge switching implementation provided by [33] we use 10 times the number of edges as the number of random edge switches. Previously performed experiments (e. g. [22]) have shown that this is enough to expect the resulting graph to be drawn uniformly at random from all graphs with the given degree sequence.

For an original graph $O = (V,E)$ with $n_o = |V|$ nodes and a desired scaling factor x, ReCoN executes the following steps:

1. Detect a community structure $\mathcal{C} = \{C_1, \ldots, C_k\}$ on O using PLM.
2. Create H as the disjoint union of x copies of O. The community structure is also copied such that the new community structure $\mathcal{D} = \{D_1, \ldots, D_{x \cdot k}\}$ consists of $x \cdot k$ communities, i. e. each copy of O gets its own copy of the community structure that is aligned with the structure of the copied graph.
3. For each community D_i, $1 \leq i \leq x \cdot k$, randomize the edges of the subgraph $H[D_i]$ that is induced by the community D_i while keeping the degree distribution using random edge switches.
4. Randomize the remaining edges, i. e. all edges in H that are not part of one of the subgraphs $H[D_i]$ using random edge switches. Note that afterwards some edges that were not in one of the $H[D_i]$ can now be inside a community. In order to avoid this, rewiring steps are performed by executing edge switches of such forbidden edges with random partners. A similar step is also used in the LFR generator where it was observed that in practice only few rewiring steps are necessary [18].

Note that it is not necessary to start with the original graph in step 3 and 4. Using any graph with the same degree sequence is enough as the result is random

anyway. Therefore, it is enough to know a community structure (as opposed to the whole original graph) and for each node the internal and external degree, i. e. how many neighbors it has inside and outside its community, respectively. For our implementation we choose this alternative. Further, we execute step 3 in parallel for all communities as the subgraphs are disjoint.

In addition to replicating important properties with high fidelity, the randomization in step 3 and 4 naturally produces random variance among the set of replicas.

4 Fitting Generative Models to Input Graphs

Parametrized generative models require fitting schemes for learning parameters from the original network. Because, usually, such schemes are not unique, exploring them would be important future work. For this study, we have chosen one scheme per model, parameters of which are summarized in Table 1 in the full version of this paper [28]. Below we discuss a fitting scheme for power law degree distributions, and briefly describe the generative models that are compared with ReCoN.

Fitting power law degree distribution (PLD). We apply our custom power law fitting scheme. A practical replication of a network requires preserving the original average (otherwise, the density will be changed) as well as minimum and maximum degrees (applications can be sensitive to such fundamental properties as degree-1 nodes and the distribution of hubs). In general, it is assumed (and implemented in many algorithms [8]) that PLD only holds starting with a minimum degree and that for smaller degrees, the distribution might be different. As the LFR generator only generates a plain PLD, we cannot apply this assumption. Therefore, we fit the PLD exponent such that, with the given minimum and maximum degree, the average degree of the real network is expected when a degree sequence is sampled from this PLD. Using binary search in the range of $[-6, -1]$, we repeatedly calculate the expected average degree until the power law exponent is accurate up to an error of 10^{-3}.

Erds–Rnyi, Barabasi-Albert, Chung-Lu and ESMC. *Erds–Rnyi* random graphs (ER) [24] are fundamental and an important baseline with the edge probability parameter that we set to produce the same edge-to-node ratio as in O. The *Barabasi–Albert* model (BA) [2] implements a preferential attachment process by which a PLD emerges, which has been claimed to be a typical feature of real complex networks. In BA, we set the number of edges coming with each new node to fit the original edge-to-node ratio. The *Chung-Lu* (CL) model [1] recreates a given degree sequence in expectation. The *Edge-Switching Markov Chain Generator* (ESMC) generates a graph that is randomly drawn from all graphs with exactly the given degree sequence (see e.g. [22], [26]). In both CL and ESMC we use the original degree sequence. To generate larger networks, x copies of this sequence are concatenated, multiplying the number of nodes by x while keeping the relative frequency of each degree.

RMAT. The *Recursive Matrix* (RMAT) model [7] was proposed to recreate various properties of complex networks, including an optional power-law degree distribution, the small-world property and self-similarity. The RMAT model can only generate

graphs with 2^s nodes, where s is an integer scaling parameter. In order to target a fixed number of nodes n_r, we calculate s so that $2^s > n_r$ and delete $2^s - n_r$ random nodes. The choice of other parameters as well as the running time of fitting are discussed in [28].

Hyperbolic Unit Disk Graphs (HUDG). The random hyperbolic graph model embeds nodes into hyperbolic geometry and connects close nodes with higher probability [17]. The unit-disk variant HUDG we use in this paper connects only nodes whose distance is below a certain threshold. We are focussing on the unit-disk variant to be able to use a very fast generator for this model [21]. The model has been shown to replicate some properties observed in real networks, such as a power-law degree distribution. This method receives as parameters the desired number of nodes, the average degree of the original network and a power law exponent which is fitted as described above. As the given power law exponent must be larger than 2, we supply at least an exponent of 2.1.

BTER. This method receives a degree distribution and the desired clustering coefficient per degree, i.e., for each degree to be realized the number of occurrences and the average clustering coefficient per degree. For scaled replicas we scale the occurrences of all degrees by the scaling factor. This leads to the target number of nodes while also preserving the general shape of the degree distribution. In order to retain the distribution of the clustering coefficients, we leave them unchanged while scaling the network.

LFR. LFR was designed as a benchmark graph generator for community detection algorithms [19]. Apart from the number of nodes it requires parameters for power law distributions of the node degrees and the community sizes, and a mixing parameter that determines the ratio between intra- and inter-cluster edges. We detect communities using PLM [30] and fit the parameters for the two power law distributions as described above using the original degree sequence and the found community sizes. The mixing parameter is set to the ratio between intra- and inter-cluster edges of the found communities. The details are described in [28].

5 Computational Experiments

Our implementations of ReCoN and the various fitting methods are based on NetworKit [31], a tool suite for scalable network analysis. It also contains many of the generators we use for comparison and provides a large set of graph algorithms we use for our experiments. NetworKit combines C++ kernels with an interactive Python shell to achieve both high performance and interactivity, a concept we use for our implementations as well. All implementations are freely available as part of the package at `https://networkit.iti.kit.edu`. This also includes a faster and parallel implementation of the LFR generator (compared to the original implementation [10]).

Our experimental platform is a shared-memory server with 256 GB RAM and 2x8 Intel(R) Xeon(R) E5-2680 cores at 2.7 GHz, using the GCC 4.8 compiler and the openSUSE 13.1 OS. More technical details are available in [28].

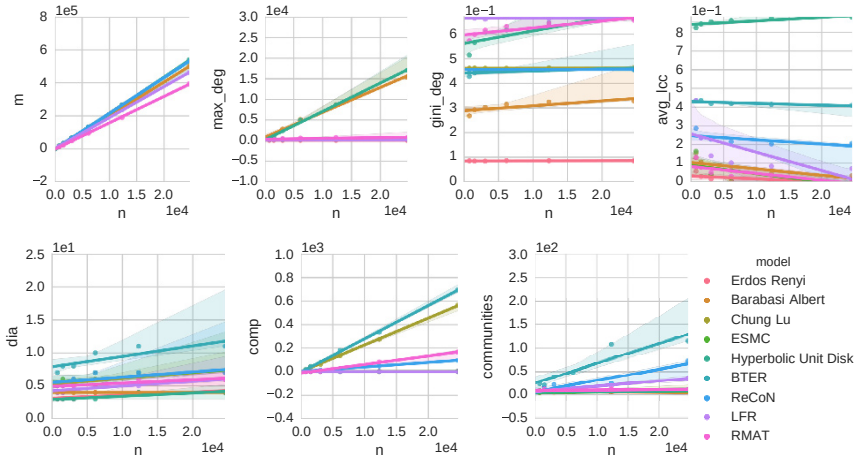

Fig. 2: Scaling behavior of the different generators on the `fb-Caltech36` network. From left to right and top to bottom: number of edges, max. degree, Gini coefficient of the degree distribution, average local clustering coefficient, diameter, number of components, number of communities.

As described in Section 2, we are interested in how well the different generators replicate certain structural features of the original networks as well as the running times of various graph algorithms. The results are described subsequently.

Scaling behavior of the generators. The following experiments consider the scaling behavior of generative models. Given the parametrization discussed before, we look at the evolution of structural features with growing scale factor x up to $x = 32$. We consider the same basic scalar features as for the real networks in Sec. 2 and, due to space constraints, point to [29] for more results.

In Figure 2, we show the results of the scaling experiments for the `fb-Caltech36` network. The number of edges of the replicas is increased almost linearly by all generators to $\approx 5 \cdot 10^5$ edges which approximately corresponds to 32 times the edges of the original network. Therefore, all generators seem to keep the average degree of the original network, which is expected as it is a parameter of all considered generators. Surprisingly, the maximum degree strongly increases up to 10 or 15 thousand with HUDG and BA generators, respectively. The original maximum degree is 248, so that the new value is even significantly higher than the scaled maximum degree (i. e. $248 \cdot 32$). Actually, from the scaling study in Sec. 2, we could expect an increase, but rather in a lower range, so the degree distribution of BA and HUDG generators are not realistic. Concerning the Gini coefficient, one can clearly see that ER does not generate a skewed degree distribution at all. All generators that get the exact degree sequence as input keep the Gini coefficient constant, which is expected and also relatively realistic from our scaling study.

The original average local clustering coefficient of 0.43 is almost exactly reproduced by BTER in which it is an input parameter. The HUDG method increases it

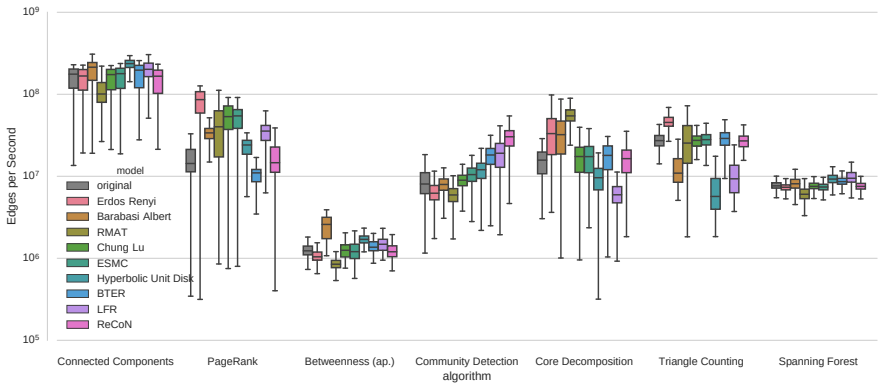

Fig. 3: Running time replication of a set of network analysis algorithms. Running times are in edges per second, i.e., higher is faster.

to 0.8, most others obtain very small values. Our new ReCoN generator is less far off with 0.25 and a slightly decreasing clustering coefficient; the latter is actually realistic as we saw in Sec. 2. LFR is able to generate a clustering coefficient above 0.2 initially. Other generators produce much lower clustering coefficients. The original diameter of 6 is almost exactly kept by ReCoN, all other generators except BTER produce networks with slightly lower diameters, while BTER generates networks whose diameter is almost twice bigger. All generators show a slight increase of the diameter when the networks are larger, which is consistent with our scaling study. While most generators produce networks with just a single connected component, CL and BTER generate a large number, RMAT and ReCoN a moderate number of connected components. In the case of CL, BTER and RMAT, this is probably due to a large number of degree-0 nodes. The original network consists of a giant component and 3 small components; ReCoN scales them linearly, which is due to its parametrization. The original network is split into eight non-trivial communities, that number should increase slowly according to Sec. 2. Only in the networks generated by BTER, ReCoN and LFR, PLM can find a significant and increasing amount of communities. While PLM finds over 100 non-trivial communities in the network generated by BTER, there are fewer communities detectable in the networks generated by ReCoN and even less in the ones generated by LFR. Overall, ReCoN is the only generator that keeps the degree distribution, and produces a realistic clustering coefficient and a small diameter while keeping the graph connected and preserving a moderate number of communities. All other generators are either unable to keep the diameter or the connectivity or the number of communities. It is part of future work to investigate whether the full hyperbolic random graph model can alleviate the weaknesses of the unit-disk case.

Replicating running times of graph algorithms. Synthetic graphs are frequently used in algorithm engineering to estimate the running time of an algorithm assuming that this time will be similar on real networks. We examine if this is indeed the case with the generative models we consider. Using the previously described generators

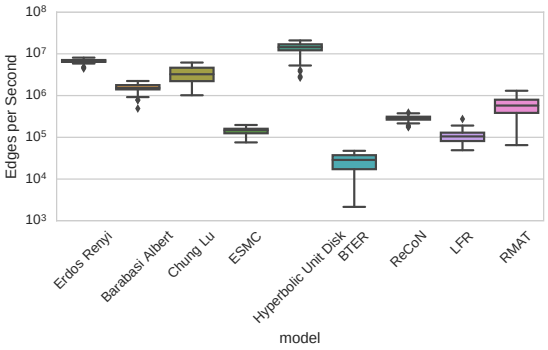

Fig. 4 Fitting and generating: processing speed measured in edges/s (size of replica graph / total running time, measured on 100 Facebook graphs)

and fitting schemes, we generate replicas of 100 Facebook networks and test a variety of graph algorithms (see Sec. 2) on both the original and replica sets.

Our experiments demonstrate (see Fig. 3) that the running times on the replica sets often do not match those on the original set. The gray segments of the box plots represent the distribution of running times measured on a set of original networks. Ideally, the distribution on the synthetic networks would be identical. The difference is statistically nontrivial, though. Small variance between the models exists for connected components and spanning forest computations, since their running time is nearly constant per edge. Other algorithms exemplify how much running time can depend on network structure, especially community detection, core decomposition, triangle counting and PageRank. In general, the running time measurements obtained on ReCoN match the originals closely in most cases. An exception is community detection, where PLM seems to profit from ReCoN's explicit model of communities. BTER shows close matches, too.

Generator running times. In Fig. 4, we show the running times of parameter fitting and generating a replica for all methods. Processing speed is given in the number of edges per second. The entire set of Facebook networks was used to produce the measurements, so generated replicas range from about 15000 to 1.5 million edges. For all models, generating the graph takes up the vast majority of time. BTER's MATLAB-based implementation is slowest, while the ER and HUDG generators are the fastest. Our implementations of LFR and ReCoN are not among the fastest generators, but fast enough to produce millions of edges in minutes.

6 Conclusion

We have presented a new generator, ReCoN, for replicating and scaling existing networks. In an extensive experimental evaluation (not all results could be shown due to space constraints, see [28, 29] for more results) we have shown that ReCoN is capable of generating networks which are (i) similar to the original network in terms of important structural measures and (ii) lead to similar running times of many graph and network analysis algorithms. Using ReCoN it is possible to realistically replicate an existing network, and to scale the synthetic version by orders of magnitude, e. g., in

order to test algorithms on larger data sets where they are not available. Furthermore, it allows to create anonymized copies of such networks that can be distributed freely and allow to conduct representative experiments on them. While other generators sometimes perform better concerning certain criteria, none of the other generators is capable of approximately reproducing such a wide range of properties and running times.

Acknowledgements This work is partially supported by the DFG under grants ME 3619/3-1 and WA 654/22-1 within the Priority Programme 1736 *Algorithms for Big Data*, and by the NSF awards #1522751 and #1647361. Funding was also provided by *Karlsruhe House of Young Scientists* via the *International Collaboration Package*.

References

[1] Aiello, W., Chung, F., Lu, L.: A random graph model for massive graphs. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing, pp. 171–180. Acm (2000)
[2] Albert, R., Barabási, A.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
[3] An, W.: Fitting ERGMs on big networks. Social Science Research **59**, 107 – 119 (2016). Special issue on Big Data in the Social Sciences
[4] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.U.: Complex networks: Structure and dynamics. Physics reports **424**(4), 175–308 (2006)
[5] Caldarelli, G., Vespignani, A.: Large scale structure and dynamics of complex networks. World Scientific (2007)
[6] Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: Proc. 4th SIAM Intl. Conf. on Data Mining (SDM). SIAM, Orlando, FL (2004)
[7] Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. Computer Science Department p. 541 (2004)
[8] Clauset, A., Shalizi, C.R., Newman, M.E.: Power-law distributions in empirical data. SIAM review **51**(4), 661–703 (2009)
[9] Dasari, N.S., Ranjan, D., Zubair, M.: ParK: An efficient algorithm for k-core decomposition on multicore processors. In: 2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014, pp. 9–16. IEEE (2014)
[10] Fortunato, S.: Benchmark graphs to test community detection algorithms. URL `https://sites.google.com/site/santofortunato/inthepress2`
[11] Geisberger, R., Sanders, P., Schultes, D.: Better approximation of betweenness centrality. In: ALENEX, pp. 90–100. SIAM (2008)
[12] Goldenberg, A., Zheng, A.X., Fienberg, S.E., Airoldi, E.M.: A survey of statistical network models. Foundations and Trends® in Machine Learning **2**(2), 129–233 (2010)
[13] Gutfraind, A., Meyers, L., Safro, I.: Musketeer: Multiscale entropic network generator. URL `https://people.cs.clemson.edu/~isafro/musketeer/index.html`
[14] Gutfraind, A., Safro, I., Meyers, L.A.: Multiscale network generation. In: 18th International Conference on Information Fusion, FUSION 2015, Washington, DC, USA, July 6-9, 2015, pp. 158–165 (2015)
[15] Hamann, M., Lindner, G., Meyerhenke, H., Staudt, C.L., Wagner, D.: Structure-preserving sparsification methods for social networks. Social Netw. Analys. Mining **6**(1), 22:1–22:22 (2016)
[16] Kolda, T.G., Pinar, A., Plantenga, T., Seshadhri, C.: A scalable generative graph model with community structure. arXiv preprint arXiv:1302.6636 (2013)

[17] Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., Boguñá, M.: Hyperbolic geometry of complex networks. Physical Review E **82**, 036,106 (2010)
[18] Lancichinetti, A., Fortunato, S.: Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Physical Review E **80**(1), 016,118 (2009)
[19] Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Physical Review E **78**(4), 046,110 (2008)
[20] Leskovec, J., Faloutsos, C.: Scalable modeling of real graphs using kronecker multiplication. In: Proc. 24th Intl. Conference on Machine learning, pp. 497–504. ACM (2007)
[21] von Looz, M., Meyerhenke, H., Prutkin, R.: Generating random hyperbolic graphs in subquadratic time. In: Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings, pp. 467–478 (2015)
[22] Milo, R., Kashtan, N., Itzkovitz, S., Newman, M.E.J., Alon, U.: On the uniform generation of random graphs with prescribed degree sequences. eprint arXiv:cond-mat/0312028 (2003)
[23] Newman, M.: Networks: an introduction. Oxford University Press (2010)
[24] P. Erdős, A.R.: On the Evolution of Random Graphs. Publication of the Mathematical Institute of the Hungarian Academy of Sciences (1960)
[25] Robins, G., Pattison, P., Kalish, Y., Lusher, D.: An introduction to exponential random graph (p*) models for social networks. Social Networks **29**(2), 173 – 191 (2007). Special Section: Advances in Exponential Random Graph (p*) Models
[26] Schlauch, W.E., Horvát, E.Á., Zweig, K.A.: 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
[27] Snijders, T.A.: The statistical evaluation of social network dynamics. Sociological methodology **31**(1), 361–395 (2001)
[28] Staudt, C., Hamann, M., Safro, I., Gutfraind, A., Meyerhenke, H.: Generating Scaled Replicas of Real-World Complex Networks. Tech. rep., arXiv (2016). URL `http://arxiv.org/abs/1609.02121`. ArXiv:1609.02121
[29] Staudt, C.L.: Algorithms and software for the analysis of large complex networks. Ph.D. thesis, Karlsruhe Institute of Technology (2016). DOI 10.5445/IR/1000056470
[30] Staudt, C.L., Meyerhenke, H.: Engineering parallel algorithms for community detection in massive networks. IEEE Trans. on Parallel and Distributed Systems **27**(1), 171–184 (2016)
[31] Staudt, C.L., Sazonovs, A., Meyerhenke, H.: NetworKit: A tool suite for large-scale network analysis. Network Science To appear
[32] Traud, A.L., Mucha, P.J., Porter, M.A.: Social structure of facebook networks. Physica A: Statistical Mechanics and its Applications **391**(16), 4165–4180 (2012)
[33] Viger, F., Latapy, M.: Random generation of large connected simple graphs with prescribed degree distribution. In: 11th International Conference on Computing and Combinatorics. Kunming, Yunnan, Chine (2005)