Journal of Statistical Software

November 2020, Volume 96, Issue 7. *doi: 10.18637/jss.v096.i07*

fastnet: An R Package for Fast Simulation and Analysis of Large-Scale Social Networks

Xu Dong
Tamr Inc.

Luis Castro
World Bank

Nazrul Shaikh
Cecareus Inc.

Abstract

Traditional tools and software for social network analysis are seldom scalable and/or fast. This paper provides an overview of an R package called **fastnet**, a tool for scaling and speeding up the simulation and analysis of large-scale social networks. **fastnet** uses multi-core processing and sub-graph sampling algorithms to achieve the desired scale-up and speed-up. Simple examples, usages, and comparisons of scale-up and speed-up as compared to other R packages, i.e., **igraph** and **statnet**, are presented.

Keywords: social network analysis, network simulation, network metrics, multi-core processing, sampling.

1. Introduction

It has been about twenty years since the introduction of social network analysis (SNA) software such as **Pajek** (Batagelj and Mrvar 1998) and **UCINET** (Borgatti, Everett, and Freeman 2002). Though these software packages are still existent, the last ten years have witnessed a significant change in the needs and aspiration of researchers working in the field. The growth of popular online social networks, such as Facebook, Twitter, LinkedIn, Snapchat, and the availability of data from large systems such as the telecommunication system and the internet of things (IoT) has ushered in the need to focus on computational and data management issues associated with SNA. During this period, several Python (Van Rossum *et al.* 2011) and Java based SNA tools, such as **NetworkX** (Hagberg, Schult, and Swart 2008) and **SNAP** (Leskovec and Sosič 2016), and R (R Core Team 2020) packages, such as **statnet** (Hunter, Handcock, Butts, Goodreau, and Morris 2008; Handcock *et al.* 2019) and **igraph** (Csardi and Nepusz 2006) have emerged that enable researchers and practitioners to perform analytic tasks on such large social networks. A more comprehensive list of such software and packages is provide in Table 1.

Initial release	Name	Contributor(s)	R version	Other version(s)	Commercial license
1997	**Pajek**	Batagelj et al.	–	Windows program	–
1997	**InFlow**	Krebs	–	Windows program	✓
1999	**UCINET** (including **NetDraw**)	Borgatti et al.	–	Windows program	–
2000	**Graphviz**	AT&T Labs Research	–	C	✓
2001	**Tulip**	Auber and Mary	–	C++	–
2001	**NetMiner**	Cyram Inc.	–	Java	✓
2002	**Cytoscape**	Institute of Systems Biology	–	Java	✓
2002	**Visone**	University of Konstanz	–	Java	–
2005	**JUNG**	Madadhain et al.	–	Java	–
2006	**igraph**	Csardi et al.	✓	Python, C++	–
2006	**Lisna**	Usher	–	Python	–
2006	**Netlytic**	Gruzd	–	JavaScript	✓
2007	**Network Workbench**	Börner	–	Java	–
2007	**Sentinel Visualizer**	FMS Advanced Systems Group	–	Windows program	✓
2008	**NetworkX**	Hagberg et al.	–	Python	–
2008	**statnet** (including **ergm**, **network**, **networksis**, **sna**, **networkDynamic**)	Handcock et al.	✓	N/A	–
2008	**SoNIA**	McFarland et al.	–	Java	–
2008	**NodeXL**	Social Media Research Foundation	–	Microsoft Excel	✓
2008	**UNISoN**	Leonard	–	Java	–
2008	**Gephi**	Bastian et al.	–	Java, OpenGL	–
2009	**SNAP**	Leskovec	–	Python, C++	–
2011	**GraphStream**	Balev et al.	–	Java	–
2012	**EgoNet**	McCarty	–	Java	–
2012	**graph-tool**	Peixoto	–	Python, C++	–
2012	**GraphChi**	Kyrola et al.	–	C++, Java, Scala	–
2013	**Networkit**	Meyerhenke	–	Python, C++	–
2013	**Linkurious**	Heymann	–	JavaScript	✓
2013	**AdvancedMiner SNA**	ALGOLYTICS	–	Java	✓
2013	**ScaleGraph**	Suzumura	–	MPI	–
2014	**SocNetV**	Kalamaras	–	C++	–
2015	**Tnet**	Opsahl	✓	N/A	–
2015	**Polinode**	Pitts	–	Cloud-based	✓
2015	**Socilyzer**	Socilyzer	–	JavaScript	✓
2017	**tidygraph**	Thomas Pedersen	✓	N/A	–

Table 1: Major SNA software, including tools and packages.

However, these software packages are seldom geared towards computationally-efficient use of available hardware resources for multi-core processing or sampling algorithms for the estimation of network metrics (Ebbes, Huang, and Rangaswamy 2016). This paper presents our new scalable R-based SNA package, called **fastnet** (Shaikh, Dong, and Castro 2020), which can simulate larger networks and analyze large networks faster. Thus, given a resource constraint such as available RAM size, researchers can simulate larger networks and analyze them faster on **fastnet** than on other SNA systems such as **NetworkX**, **statnet**, and **igraph**. **fastnet** is available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=fastnet.

fastnet achieves a speed-up and scale-up by adopting the parallel computing functionalities provided by **foreach** and **doParallel** (Kane, Emerson, and Weston 2013; Weston 2019, 2020) and managing work allocation to all the available computational resources[1]. Though **fastnet** is not the first R-based SNA package that uses parallel processing, the algorithms in **fastnet** are customized to manage and leverage distributed computing across multiple cores. The detail of how to utilize and customize parallel processing in **fastnet** can be found in Section 2.4.

fastnet also enables the speed-up on the computational time required to estimate the structural metrics of networks. **fastnet** combines multi-core processing with sampling-based approaches that approximate the network metrics. The speed-up and the error bounds of the estimates of the network metrics can be controlled by the sample size and the number of core available for multi-core processing. The methods used in **fastnet** are either random node sampling, random link sampling, or random node pair sampling, allowing for easy implementation and transparent measurements of errors. In the following subsections, we will present a brief introduction of the package history, the functionalities embedded, and point to some key notations for understanding the theoretical background, and the form of network representation used in this package.

1.1. Package history

The **fastnet** package was created for the purpose of studying the role played by the structure of social networks on diffusion dynamics, such as in new product diffusion, rumor propagation, and disease spread. During the study, we were confronted with the repeated task of simulating large networks that emulate real social networks and measuring the corresponding network metrics. By introducing the idea of multi-core processing and sampling, we started to create our own R functions trying to simulate large graphs[2] using standard algorithms (such as those proposed by Erdős and Rényi (1960), Watts and Strogatz (1998), Barabási and Albert (1999), and Newman, Strogatz, and Watts (2001) that are widely applied in social science studies) as well as network metrics that characterize a social network (such as degree distribution, mean local clustering coefficient and APL)[3]. The first version of **fastnet** package was released

[1]The other way to address this limitation of a regular PC is to implement high performance computing techniques, such as distributed computing (Liu, Zhang, and Yan 2010) and cloud computing (Yu *et al.* 2012). However, since these techniques require specialized knowledge of programming and financial costs for the extra infrastructure, it is beyond the scope of **fastnet**, which is developing an open-source SNA tool for large-scale social networks that can be operated on existing computational devices.

[2]In this paper, we interchangeably use the terms "network" and "graph", "link" and "edge", as well as "node" and "agent".

[3]There are a few network algorithms that are unable to be processed in parallel. For example, the generation of Barabási-Albert graphs has a preferential attachment mechanism that is unable to be computationally-parallelized. We implemented such algorithms sequentially.

in December 2016 on the Comprehensive R Archive Network (CRAN). As far as we know, **fastnet** is the first SNA package that aims at fast simulating large-scale social networks and calculating network metrics within the R environment when there is a constraint on the available computational resources.

1.2. Functionality

At present, the **fastnet** package includes over 40 functionalities for SNA. These functionalities include the ability to:

- Simulate regular networks, such as complete graphs, ring lattices, and connected caveman networks.[4]
- Simulate random networks through various stochastic processes and classic generating algorithms, such as two variants of *Erdős-Rényi* random graphs, i.e., $g(N, p)$ and $g(N, m)$, Barabási-Albert graphs, and Watts-Strogatz graphs. Erdős-Rényi and Barabási-Albert graphs can be generated in both direct or indirect forms.
- Simulate random networks using cluster centric algorithms such as the Holme-Kim, Cluster-Affiliation, and Degree-constraint graphs.
- Convert network represented as **igraph** objects, adjacency matrices, and edgelists, to become **fastnet** objects.
- Compute degree-related network metrics, such as mean degree, median degree, and degree entropy.
- Compute distance-related network metrics, such as APL, diameter and average eccentricity.
- Compute cluster-related network metrics, such as global transitivity and mean local clustering coefficient.
- Compute eigenvector-related network metrics, such as mean eigenvector centrality.
- Compute other network metrics, such as graph density.
- Obtain the degree distribution of a network.
- Obtain the neighborhood list of any given node.
- Draw plots, such as the histogram of the degree distribution, the cumulative degree distribution of network.

The core functionalities and their corresponding call names and parameters are listed in Table 2 and Table 3 .

1.3. Some notations

A network is simply a collection of nodes and edges, formally denoted as $G = (V, E)$, where V is the node set and E is the edge set. In SNA, nodes usually represent individuals or actors who perform social activities, while edges usually represent social interactions or relations existing between a pair of nodes. The size of a network is the number of nodes it includes, denoted as $|V|$, where $|\cdot|$ is the cardinality operator. The network can be directed or undirected; if the elements in E are fully paired, where each pair includes two edges between node i and node j, with one originated from node i to node j and another from node j to node i, the

[4]Note that the latter two regular networks are performed as base graphs of Watts-Strogatz networks and rewired caveman networks, respectively, with a stochastic rewiring process being embedded.

Function	Call name	Parameters
Barabási-Albert power-law graph	`net.barabasi.albert`	Network size, Connection number
Caveman graph	`net.caveman`	Number of cliques, Number of nodes per clique
Complete graph	`net.complete`	Network size
Erdős-Rényi random graph g(n, p)	`net.erdos.renyi.gnm`	Network size, Connecting probability
Erdős-Rényi random graph g(n, m)	`net.erdos.renyi.gnm`	Network size, Number of edges
Holme-Kim graph	`net.holme.kim`	Network size, Connection number, Triad formation probability
Random graph with a power-law degree distribution that has an exponential cutoff	`net.random.plc`	Network size, Exponential cutoff of the degree distribution, Exponent of the degree distribution
Rewired caveman graph	`net.rewired.caveman`	Number of cliques, Number of nodes per clique, Link rewiring probability
Ring lattice	`net.ring.lattice`	Network size, Number of edges per node
Watts-Strogatz small-world graph	`net.watts.strogatz`	Network size, Number of edges per node, Rewiring probability
Degree-Constraint graph	`net.degree.constraint`	Network size, cluster-size distribution scaling parameter
Cluster-Affiliation graph	`net.cluster.affiliation`	Network size, minimal number of membership, the scaling parameters and exponential cutoff of the membership distribution, the minimal number and the exponential cutoff of the clique-size distribution

Table 2: Network simulation algorithms implemented in **fastnet**.

Function	Call name	Parameters
Max degree	`metric.degree.max`	Input network
Effective max degree	`metric.degree.max.efficient`	Input network, Effective rate
Quantile of degree distribution	`metric.degree.effective`	Input network, Quantile value
Min degree	`metric.degree.min`	Input network
Mean degree	`metric.degree.mean`	Input network
Median degree	`metric.degree.median`	Input network
Degree entropy	`metric.degree.entropy`	Input network
Standard deviation of all degrees	`metric.degree.sd`	Input network
Diameter	`metric.distance.diameter`	Input network, Confidence level, Sampling error
Effective diameter	`metric.distance.diameter`	Input network, Confidence level, Sampling error, Effective rate
APL (Mean shortest path length)	`metric.distance.apl`	Input network, Confidence level, Sampling error, Effective rate
MPL (Median shortest path length)	`metric.distance.mpl`	Input network, Confidence level, Sampling error, Effective rate
Mean eccentricity	`metric.distance.meanecc`	Input network, Sampling probability
Median eccentricity	`metric.distance.medianecc`	Input network, Sampling probability
Global clustering coefficient	`metric.cluster.global`	Input network
Mean local clustering coefficient	`metric.cluster.mean`	Input network
Median local clustering coefficient	`metric.cluster.median`	Input network
Mean eigenvalue of the adjacency matrix	`metric.eigen.mean`	Input network
Median eigenvalue of the adjacency matrix	`metric.eigen.median`	Input network

Table 3: Network metric estimation algorithms implemented in **fastnet**.

corresponding network is called undirected; otherwise, it is called directed. For a directed network, the neighbors of node i is the set of nodes receiving edges from i, while for an undirected network the neighbors of node i is the full set of nodes that link to i. The **fastnet** package focuses on simulating simple graphs, both directed and undirected, that have no looping edges over one node, or multiple edges existing between one node and another.

Network metrics are descriptive indices of networks. Network metrics can be categorized into two groups: global and local level metrics (Dong, Rangaswamy, and Shaikh 2016). Global level network metrics capture the overall characteristics of entire network, while the local level network metrics measure individual characteristics of a node or a link on a network. Different network metrics allow users to assess both the micro and macro level properties of a target network, and tell user how close the network structures are from the topological perspective. The **fastnet** package provides both global and local level metrics of networks.

1.4. Network representation form

From the perspective of developing an effective and practical SNA tool, the problem of representing a network involves economical storage and efficient retrieval of network information. Traditionally in SNA, network information is represented by an $N \times N$ 0–1 adjacency matrix $\mathbf{A}$, in which, the element a_{ij} represents the existence of a link starting from nodes i to j. In this case, the number of memory units scale quadratically with N.[5] Taking into account that most social networks are sparse, allocating $N \times N$ memory units to store the linkage information can be uneconomical. A solution for that is to present a network as a sparse matrix, including the compressed sparse row (CSR) or the compressed sparse column (CSC) formats (Koenker and Ng 2019). These sparse matrix representations replace the use of an $N \times N$ adjacency matrix with three (one-dimensional) vectors, with one vector containing the nonzero values, one vector containing the extents of rows/columns, and one vector storing the column/rows indices. Significant efficiency gains in memory usage can be achieved by representing the adjacency matrix as a sparse matrix[6]. Also, sparse matrix representation allows faster row access and vector multiplication of matrices. However, it is not practical to use sparse matrix forms for representing a social network, since a sparse matrix form is unable to directly retrieve the most critical information that we seek to extract from a social network, which relates to two questions: 1) Who do the nodes represent? and 2) Who do the nodes link to?

Driven by the above mentioned reasons, we use a list-based egocentric representation of the network, wherein the focus is on directly presenting nodes and their connections. By using a list with size N, where each entry represents one node in a network and the size of each entry is dictated by the exact number of connections for this node, the neighbors' IDs of all nodes in the network are sorted and saved. If the average number of connections is K^-, the memory requirement for the list scales linearly with $N \times K^-$.[7] Since for most real-world social networks, K^- is significantly lower than N, the number of memory units required to store

[5]The number of links can reach up to $O(N^2)$ in a network of size N.

[6]Butts (2008a) used a sparse matrix-based representation of large and sparse networks in the **network** package, indicating a 99.8% reduction in memory requirements when a network with 100,000 links and 100,000 nodes is represented by a sparse matrix.

[7]Note that for some networks, such as Erdős-Rényi $g(N, p)$ random networks, where $K^- = N \times p$ and p is a constant representing the connecting probability, the memory requirement for the list can be converted to scale linearly with N^2.

information associated with the egocentric list is significantly smaller than the ones with the entire adjacency matrix, which scales quadratically with N.

An alternative to our egocentric representation approach is to store network information by using an array of size $N \times K^{=}$ (where $K^{=}$ is the maximum number of connections that an agent could possibly possess in the network). Though in the R environment an object with a fixed size is usually more memory efficient than an object with a flexible size (Visser, McMahon, Merow, Dixon, Record, and Jongejans 2015), the memory-efficiency, in this case, depends on the ratio between $K^{=}$ and K^{-}. The list form is more efficient when the ratio between $K^{=}$ and K^{-} is much larger than 1, which is the case for most social networks (Newman 2002). Hence, the list-based egocentric representation form better fits social network contexts (Castro, Dong, and Shaikh 2018). In sum, compared to other existing SNA tools, **fastnet** has the following advantages:

- The required memory of storing a network instance is linear in the number of nodes in a network, which is ideal for the scaling-up of network objects in the R environment.
- Lower computational time is required for the calculation of most of the network metrics, due to embedded multi-core processing and sampling strategies.

Section 2 provides some simple examples to demonstrate the usage of the functionalities in **fastnet**. The details of how to obtain network metrics that incorporate sampling-based techniques are specified. A method for network metric visualization is also specified. Section 3 provides a comparison of the speed-up and scale-up achieved using **fastnet** as compared to **igraph** and **statnet**. Section 4 uses a case study to illustrate a general process to analyze the structural properties of real social networks. The last section provides a summary of **fastnet**.

2. Working with fastnet

In this section, we try to highlight some key functions in **fastnet** and help its users gain a better understanding of the package. Simple examples are presented to illustrate the functionalities provided by the package. For additional details and usage, users can refer to the package reference manual.

2.1. Network generation

fastnet can generate random networks through a variety of standard mechanisms, such as uniform connecting, preferential attachment, edge rewiring, etc. Currently, **fastnet** provides 12 network generating algorithms, where all network generation functions start with a prefix `net`. These include `net.erdos.renyi.gnp` and `net.erdos.renyi.gnm` that generate Erdős-Rényi random networks where the degrees of nodes follow a Binomial`(n, p)` distribution, or the total number of edges is a constant `m`, `net.barabasi.albert` that generates Barabási-Albert random networks using a preferential attachment mechanism from Albert and Barabási (2002), and `net.watts.strogatz` that generates a small-world network according to Watts and Strogatz (1998). For Erdős-Rényi random networks and Barabási-Albert random networks, **fastnet** allows users to select whether the network is directed or undirected[8].

[8]By default, all networks generated by **fastnet** are undirected.

Note that the network generating functions above are also implemented in other R packages, such as **igraph** and **statnet**. However, the major difference is that **fastnet** network generation functions produce a list-based egocentric representations of a network. Other than these standard network generation functions, **fastnet** includes some other functions for generating simulated social networks. These include `net.rewired.caveman` that generates rewired caveman networks according to Watts (1999), `net.holme.kim` that generates Holme-Kim networks from Holme and Kim (2002), and `net.random.plc` that generates random networks with a power-law degree distribution and an exponential degree cutoff from Newman *et al.* (2001). `net.degree.constraint` that generates social networks that have any pre-defined degree distributions, and `net.cluster.affilication` that generates social networks that have densely-connected and overlapped communities (Dong *et al.* 2016).

Besides the generation of random networks, for the purpose of benchmarking[9], **fastnet** also generates regular networks, such as `net.complete` that generates fully-connected complete networks, `net.ring.lattice` that generates k-regular ring lattices and `net.caveman` that generates connected caveman networks according to Watts (1999).

To begin with, we need to load the **fastnet** package and setup a random seed for the purpose of reproducibility.

```
R> library("fastnet")
R> set.seed(99)
```

For instance, if we want to generate a small rewired caveman network, which has 4 clusters with 7 nodes in each, and an edge rewiring probability of 0.1, we can type the following syntax after the R prompt:

```
R> recave <- net.rewired.caveman(nc = 4, m = 7, p = 0.1)
```

fastnet helps users to create the required network with a given name `recave`. We may look at the structure of `recave` by using the `preview.net` as follows:

```
R> preview.net(recave)

[[1]]
[1] 2 3 4 5 6 7

[[2]]
[1] 3 4 5 6 7 1

[[3]]
[1]  4 5 6 7 1 2

[[4]]
[1]  5 6 7 1 2 3

[[5]]
```

[9]Here benchmarking refers to measuring how the linkage randomization of a network impacts its topological characteristics.

```
[1] 7 1 2 3 4

[[6]]
[1] 7 1 2 3 4

[[7]]
[1] 1 2 3 4 5 6

[[8]]
[1] 10 11 12 13

[[9]]
[1] 10 12 13 14

[[10]]
[1] 11 12 14  8  9
```

Here, `preview.net(recave)` shows the first ten (which is set by default) agents' neighbors information. For example, the agent with ID "2" connects to agents with IDs "1", "3", "4", "5", "6", and "7", respectively. Also, we can check the neighbor's information of any given agent ID. For instance, the neighbor's information of agent with ID "20" can be obtained as follows:

```
R> get.neighbors(recave, 20)

[1] 21 17 18 19 15 16
```

As noted above, we also generate regular networks for comparisons and benchmarks. For instance, the base connected caveman network can be generated as follows:

```
R> cave <- net.caveman(m = 4, k = 7)
```

We can compare their structures in Figure 1. Similarly, we can generate other types of networks according to the same paradigm.

2.2. Conversion from and to other network representation formats

fastnet is open to many standard network representation formats, such as edgelists and adjacent matrices, as well as some specific network objects defined by other network packages, such as **igraph** and **statnet**. In order to enable all the functionalities provided by the package, we recommend **fastnet** users to convert different kinds of data formats into **fastnet** objects, using the format conversion functions built in the package. For example, the following example uses `from.adjacency` to convert `adj_mat`, the adjacency matrix of a network instance, into `g_from_adj_mat`, a **fastnet** object of its equivalent, while `from.igraph` can transform an **igraph** object `net.igraph` into a **fastnet** object of its equivalent `g_from_igraph`. After the conversions, all the network metric calculation functions can be applied to the network objects.

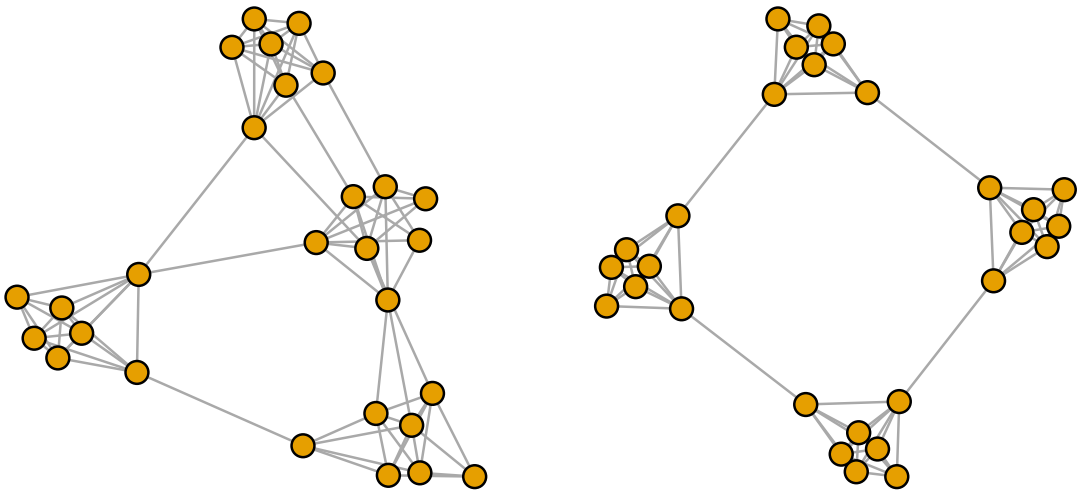

Figure 1: A rewired caveman network with 4 caves and an edge rewiring probability of 0.1 (left) and the corresponding connected caveman network (right).

```
R> adj.mat <- matrix(c(0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),
+    nrow = 4, ncol = 4)
R> g_from_adj_mat <- from.adjacency(adj.mat)
R> library("igraph")
R> net.igraph <- erdos.renyi.game(100, 0.1)
R> g_from_igraph <- from.igraph(net.igraph)
```

fastnet does not provide internal support for networks represented using sparse matrix formats such as CSR and CSC. A network object using a sparse matrix format needs to be transformed into an egocentric list format before it can be used within **fastnet**. Alternately, the network object that uses a sparse matrix representation could be transformed into its dense equivalent using an R package such as **SparseM** and then be converted to a **fastnet** object using the function `from.adjacency`. Additionally, **fastnet** also allows users to convert a **fastnet** object to equivalent network representation formats, such as an **igraph** object or an edgelist.

2.3. Network metrics calculation

Estimating network metrics in short time durations is the second key functionality of **fastnet**. In **fastnet**, all metric calculation functions start with a prefix `metric`.

Presenting degree distribution

Degree distribution is a critical metric for networks. For example, a Barabási-Albert network has a power-law degree distribution; and an Erdős-Rényi random network has a Binomial degree distribution (Newman *et al.* 2001). One of the advantages of the list-based egocentric network representation of **fastnet** is an easy way to obtain the network's degree distribution. For instance, let us first generate an Erdős-Rényi random network and present its degree distribution.

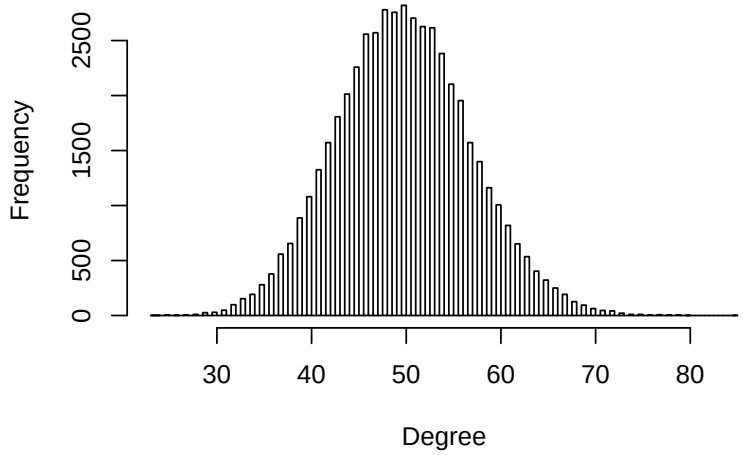

Figure 2: The histogram represents the degree distribution of the network object `uer`.

```
R> uer <- net.erdos.renyi.gnp(n = 50000, p = 0.001, d = FALSE)
R> uer.deg <- degree.collect(uer)
R> preview.deg(uer)

[1] 53 47 54 45 47 57 49 48 51 47
```

Using the first syntax of `net.erdos.renyi.gnp`, **fastnet** generates an Erdős-Rényi random network with 50,000 nodes and a connecting probability of 0.001, and saves it in a list, called `uer`. The syntax `degree.collect(uer)` obtains the degrees of all nodes in `uer`. The last syntax, `preview.deg(uer)`, shows the degrees of the first ten nodes. The histogram of the degree distribution can be displayed by using `degree.hist(uer)` (see Figure 2).

Calculating other degree-related metrics

One of the benefits for using egocentric network representation is that the users of **fastnet** can readily obtain the degree distribution of the generated network. Moreover, we can calculate other degree-related metrics. For example, we use the following syntax to calculate six major degree-related metrics:

```
R> metric.degree.max(uer)

[1] 84

R> metric.degree.min(uer)

[1] 24

R> metric.degree.sd(uer)

[1] 7.06991
```

```
R> metric.degree.mean(uer)

[1] 50.04296

R> metric.degree.median(uer)

[1] 50

R> metric.degree.effective(uer)

[1] 59
```

Here, `metric.degree.max` provides the maximum degree of `uer`; `metric.degree.min` provides the minimum degree of `uer`; `metric.degree.sd` provides the standard deviation of all degrees; `metric.degree.mean` provides the mean degree of `uer`; `metric.degree.median` provides the median degree of `uer`; and finally, `metric.degree.effective` provides the effective degree of `uer`, where the effective degree of a network is defined as the 90%-quantile of the corresponding degree distribution.

Calculating distance-related metrics

Distance-related metrics, including APL and diameter, are the most well-known global-level network metrics. **fastnet** outperforms other network analysis tools, especially in the area of efficiently calculating distance-related metrics. As far as we know, it is the first network analysis tool that introduces the idea of node-pair sampling to facilitate the processes for approximating distance-related metrics. Moreover, **fastnet** allows users to self-define the power and/or error rate, making it more manageable for users to strike a balance between result precision and time expenditure. For example, the APL of `uer` can be calculated by the function `metric.distance.apl` as follows:

```
R> metric.distance.apl(uer, full.apl = FALSE)

[1] 3.053309

R> metric.distance.apl(uer, full.apl = TRUE)

[1] 3.03678
```

If parameter `full.apl` is set to be `FALSE`, the associated random node-pair sampling algorithm (Castro and Shaikh 2018) is invoked with a default confidence level at 0.95 and the error rate level at 0.03. Note that, by setting `full.apl` to be `TRUE`, **fastnet** allows users to avoid the sampling process in order to obtain the exact APL of the network, but it will take longer time compared with the sampling method (Brandes and Pich 2007). Similar to the aforementioned process, **fastnet** provides functions to calculate a network's median path length (Watts 2003), diameter (Newman 2010) and effective diameter (Palmer, Siganos, Faloutsos, Faloutsos, and Gibbons 2001), mean eccentricity (West 2001) and median eccentricity (West 2001), which are demonstrated as follows:

```
R> metric.distance.mpl(uer)

[1] 3

R> metric.distance.diameter(uer)

[1] 4

R> metric.distance.effdia(uer, effective_rate = 0.9, p = 0.1)

[1] 3

R> metric.distance.meanecc(uer, p = 0.1)

[1] 4

R> metric.distance.medianecc(uer, p = 0.1)

[1] 4
```

Calculating cluster-related metrics

Cluster-related metrics are a group of metrics that characterize the clustering effect of a network. **fastnet** provides three cluster-related metrics functions: `metric.cluster.global` provides the global clustering coefficient (Luce and Perry 1949), or referred to as global transitivity (Wasserman and Faust 1994); `metric.cluster.mean` provides the mean local clustering coefficient (Watts and Strogatz 1998); and `metric.cluster.mean` provides the median local clustering coefficient (Uetz *et al.* 2006). For example, here we calculate network `uer`'s cluster-related metrics as follows:

```
R> metric.cluster.global(uer)

[1] 0.001010511

R> metric.cluster.mean(uer)

[1] 0.001009568

R> metric.cluster.median(uer)

[1] 0.0008865248
```

As we can see, all three cluster-related metrics of `uer` are extremely small, which matches the theoretical low-clustering property of Erdős-Rényi random networks (Newman 2003).

2.4. Multi-core processing

fastnet speeds up generation of networks and calculation of network metrics by utilizing multi-core processing on a single machine. We invoke multi-core processing by using the **doParallel** and **foreach** packages provided by R and systematically allocate the available cores to the pending tasks by parallelizing the calculation and/or simulation into multiple small non-overlap tasks. **fastnet** has parallelized the distance- and cluster-based metric calculation functions, as well as all non-evolutionary synthetic network generation functions. To identify what functions can be processed in a parallel way, a user may check if the function contains the argument `ncores` or not. Specifically, users can tune the argument `ncores` to specify how many cores are invested in the computation. By default, **fastnet** uses all available cores in the machine by setting `ncores` as `detectCores()`, which detects how many cores are available. For example, a user can generate an Erdős-Rényi random network using the multi-core processing function `net.erdos.renyi.gnp(n, p, ncores)` provided in **fastnet**. The following code generates three Erdős-Rényi random networks with the same size and connectivity probability ($n = 10,000, p = 0.01$), where the number of cores used are different.

```
R> g_2 <- net.erdos.renyi.gnp(10000, 0.01, ncores = 2, d = TRUE)
R> g_4 <- net.erdos.renyi.gnp(10000, 0.01, ncores = 4, d = TRUE)
R> g_all <- net.erdos.renyi.gnp(10000, 0.01,
+    ncores = parallel::detectCores(), d = TRUE)
```

Note that a few network metric calculations or synthetic network simulations do not require multi-core processing to scale-up or speed-up, and not all computational processes can be outsourced to the R packages **doParallel** and **foreach** due to their limitations on parallelizing matrix operations and/or evolutionary algorithms.

2.5. Network visualization

fastnet provides a `draw.net` function that re-masks the plotting function from **igraph** in order to plot **fastnet** network objects and the associated degree information. The development and implementation of parallel algorithms for network visualization have been left out of the scope for the current version of **fastnet**.

Topology visualization

We use `draw.net` to plot a ring lattice and a Barabási-Albert network, for the purpose of visual comparison (see Figure 3):

```
R> rl <- net.ring.lattice(n = 12, k = 4)
R> ba <- net.barabasi.albert(n = 12, m = 4)
R> par(mfrow = c(1, 2))
R> draw.net(rl)
R> draw.net(ba)
```

Figure 3: Visualization of a ring lattice and a Barabási-Albert network (both with 12 nodes).

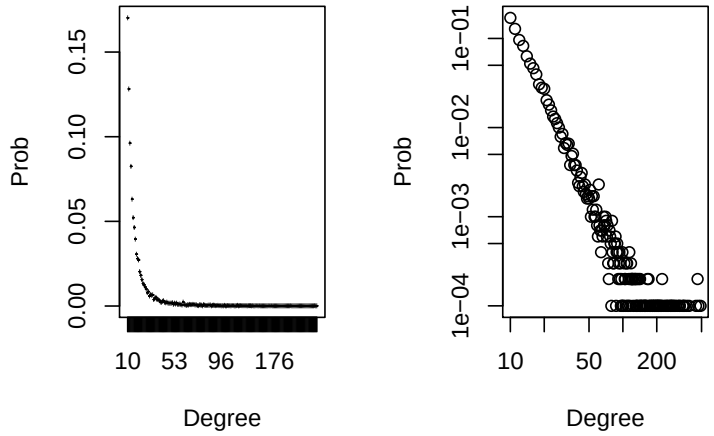

Figure 4: Visualization of the degree distribution of a Barabási-Albert network, one with standard scale and one with a logarithm scale on both `x`- and `y`-axis.

Degree distribution visualization

fastnet also provides visualization tools to plot the standard and cumulative empirical degree distribution for further analysis. Though we can simply apply the `degree.hist` function on a degree distribution object to obtain the degree histogram of a network, it is not a precise presentation of the characteristics of degrees for most cases. Instead a typical way to plot the degree distribution is to lay out the occurrence frequency of each degree. **fastnet** enables the plotting of two types of degree distributions of a network: one with standard scale and one with a logarithm scale on both the `x`- and `y`-axis.

```
R> ba.large <- net.barabasi.albert(n = 10000, m = 10)
R> par(mfrow = c(1, 2))
R> degree.dist(ba.large, cumulative = FALSE, log = FALSE)
R> degree.dist(ba.large, cumulative = FALSE, log = TRUE)
```

In the above example, we firstly generate a Barabási-Albert network with `n = 10,000` and `m = 10` and save it as a **fastnet** object called `ba.large`. The left-hand side plot displays the standard degree distribution of `ba.large` and the right-hand side plot shows the logarithm-scale degree distribution of `ba.large` (see Figure 4). The other variation of the standard degree distribution is the cumulative degree distribution. **fastnet** provides two forms of the

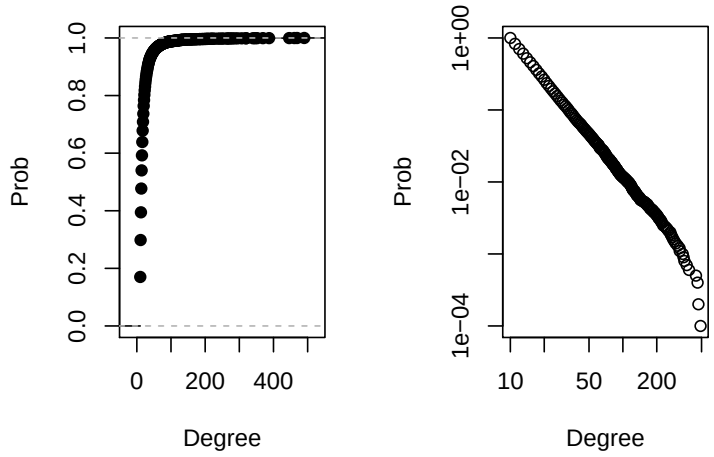

Figure 5: Visualization of two types of cumulative empirical degree distributions (type (1), left; type (2), right).

cumulative degree distribution, type (1) and type (2), as follows:

$$c(d) = \sum_{i \leq d} p(i), \tag{1}$$

$$c(d) = \sum_{i \geq d}^{d_{\max}} p(i), \tag{2}$$

where $p(i)$ is the proportion of nodes with degree i and $d_{\max}$ is the maximum degree in a network. For example, to draw the cumulative empirical degree distribution of the same Barabási-Albert network in the previous example, we can use the following syntax:

```
R> par(mfrow = c(1, 2))
R> degree.dist(ba.large, cumulative = TRUE, log = FALSE)
R> degree.dist(ba.large, cumulative = TRUE, log = TRUE)
```

In Figure 5, the plot on the left-hand side displays the type (1) cumulative degree distribution; and the plot on the right-hand side displays the type (2) cumulative degree distribution. We find that both cumulative degree distributions of the Barabási-Albert network presented in Figure 5 follow a power-law distribution, which validates a fundamental theoretical inference of a Barabási-Albert network (Barabási and Albert 1999).

3. Comparing with igraph and statnet

We provide a simple comparison of the speed-up and scale-up achieved using standard Erdős-Rényi graphs. We use the same data but different packages, i.e., **fastnet** (version 0.1.4), **igraph** (version 1.1.2) and **sna** (Butts 2008b; version 2.4) from the **statnet** suite. The speed-up and scale-up achieved while generating Erdős-Rényi networks of various sizes are presented in Figures 6 and 7, respectively. The speed-up achieved in the approximation of the APL of Erdős-Rényi networks is presented in Figure 8. All computations were done on a Dell Precision Tower Workstation (T7910) with Intel Xeon CPU E5-2637 v3 @ 3.50GHz (8 cores) and 128GB RAM memory. The operating system is Microsoft Windows Server 2012 R2 Datacenter.

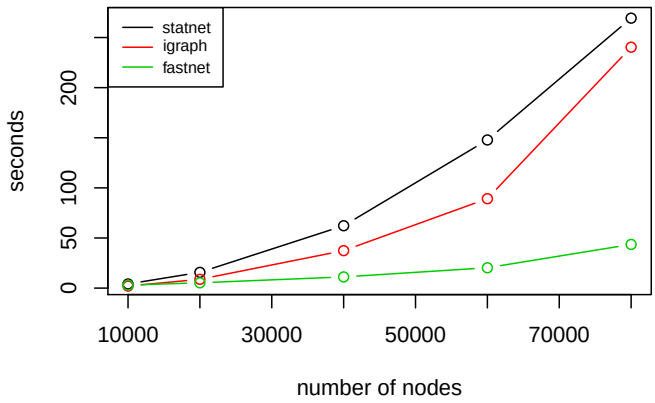

Figure 6: The time elapsed for generating directed Erdős-Rényi networks with various sizes and a fixed connecting probability (`p = 0.05`) using **fastnet**, **igraph** and **statnet**.

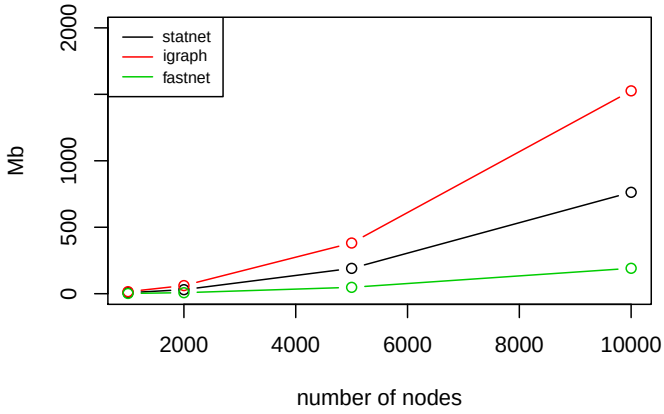

Figure 7: The memory requirements for generating undirected Erdős-Rényi networks with various sizes and a fixed connecting probability (`p = 0.5`) using **fastnet**, **igraph** and **statnet**.

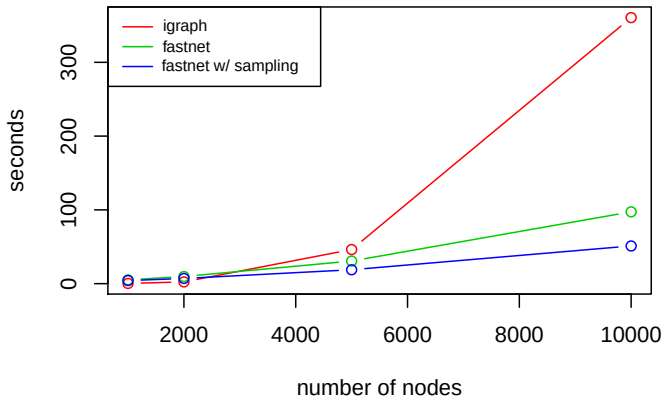

Figure 8: The time elapsed for calculating APL for undirected Erdős-Rényi networks with various sizes and a fixed connecting probability (`p = 0.5`) using **igraph** and **fastnet** (both with and without sampling strategy). Note that **statnet** is unable to calculate the APL for a network with over 1,000 nodes in this setting.

4. Case study

The case study presented in this section has been designed to illustrate the use of **fastnet** for analyzing real social networks. We use the Twitter network data from the Stanford network analysis project (SNAP; Leskovec and Krevl 2014). The network contains 81,306 nodes. Since the edgelist of the network is stored in a `.txt` file, we first use `read.table` to read it in R.

```
R> data <- read.table("twitter_combined.txt")
```

Next, we use `from.edgelist` to import the data as a **fastnet** object `twitter.real`. The functions `length(twitter.real)` and `length(unlist(twitter.real))` can be applied here to obtain the number of nodes and the number of edges in `twitter.real`, respectively.

```
R> twitter.real <- from.edgelist(data)
R> n <- length(twitter.real)
R> e <- length(unlist(twitter.real))
```

All functions described in Section 2 can now be applied to the **fastnet** object `twitter.real`. For example, we can estimate the network metrics[10].

```
R> metric.distance.apl(twitter.real, full.apl = FALSE)
R> metric.distance.apl(twitter.real, full.apl = TRUE)
R> metric.cluster.mean(twitter.real)
```

We can simulate synthetic networks that preserve some characteristics of the real network and compare them to `twitter.real`. For example, we can use `net.erdos.renyi.gnm` to generate a $g(N, m)$ Erdős-Rényi random network, where the two imperative arguments are the number of nodes and the number of edges. Similarly, we can use `net.barabasi.albert` to generate a Barabási-Albert random network, where the two imperative inputs are the number of nodes and the number of edges added when a new node is introduced to the generated graph.

```
R> twitter.er <- net.erdos.renyi.gnm(n, e)
R> twitter.ba <- net.barabasi.albert(n, round(e/(2*n)))
```

In order to compare the structural differences of the synthetic networks and the real network, we can again resort to use the network metrics such as mean degree, average path length, and global clustering coefficient, and measure the gaps between the actual versus predicted.

```
R> er.mean.degree.error.rate <- (metric.degree.mean(twitter.er) -
+    metric.degree.mean(twitter.real))/metric.degree.mean(twitter.real)
R> ba.mean.degree.error.rate <- (metric.degree.mean(twitter.ba) -
+    metric.degree.mean(twitter.real))/metric.degree.mean(twitter.real)
R> er.apl.error.rate <- (metric.distance.apl(twitter.er) -
+    metric.distance.apl(twitter.real))/metric.distance.apl(twitter.real)
R> ba.apl.error.rate <- (metric.distance.apl(twitter.ba) -
+    metric.distance.apl(twitter.real))/metric.distance.apl(twitter.real)
```

[10]It took about 42 seconds on our 16-core, 128GB RAM workstation for running `metric.distance.apl(twitter.real, full.apl = FALSE)`; about 360 seconds for running `metric.distance.apl(twitter.real, full.apl = TRUE)`.

```
R> er.gcc.error.rate <- (metric.cluster.global(twitter.er) -
+    metric.cluster.global(twitter.real))/
+    metric.cluster.global(twitter.real)
R> ba.gcc.error.rate <- (metric.cluster.global(twitter.ba) -
+    metric.cluster.global(twitter.real))/
+    metric.cluster.global(twitter.real)
```

Using the codes for calculating the error rates of the predicted values, we find that the error rates of mean degree predicted by the Erdős-Rényi and the Barabási-Albert graphs are 0.00% and 0.72%, respectively; the error rates of average path length predicted by the Erdős-Rényi and the Barabási-Albert graphs are 21.92% and 24.66%, respectively; the error rates of global clustering coefficient predicted by the Erdős-Rényi and the Barabási-Albert graphs are 99.14% and 97.32%, respectively.

5. Closing comments

The growth of online social networks, such as Facebook, and Twitter, and the availability of data from large systems such as the Internet and telecommunication networks has ushered in the need to focus on computational and data management issues associated with SNA. Researchers and practitioners are therefore seeking advanced computational tools to simulate network structures and analyze existing networks. **fastnet** aims at catering to this demand. **fastnet** is computational efficient and can generate an egocentric form of networks using various standard network generation algorithms. **fastnet** allows users to import and export objects from and to other packages in R, such as **igraph** and **statnet**, and offers the users the opportunity to speed-up the computations of network metrics by using a combination of sampling-based algorithms and multi-core processing. It is an excellent tool for users who do not have access to large computational facilities and are using PCs with multi-core processors for SNA.

Admittedly, as an alternative to utilizing high-performance computing, **fastnet** is unable to ultimately solve the large-scale network analysis issue, due to the inborn limitation of PCs (i.e., constrained CPU and memory deployed for saving spaces and energy). Hence, the next step for future **fastnet** developers is to create an R-based API to let **fastnet** easily get access to advanced computing strategies, such as distributed computing and/or cloud computing in order to expand its analytic capability.

Acknowledgments

The authors would like to thank Christian Llano Robayo, who has contributed to refining details of **fastnet**. This research was partially supported by the United States Office of Naval Research (ONR) through grant N00014-10-1-0140.

References

Albert R, Barabási AL (2002). "Statistical Mechanics of Complex Networks." *Reviews of Modern Physics*, **74**(1), 47–97. doi:10.1103/revmodphys.74.47.

Barabási AL, Albert R (1999). "Emergence of Scaling in Random Networks." *Science*, **286**(5439), 509–512. doi:10.1126/science.286.5439.509.

Batagelj V, Mrvar A (1998). "**Pajek** – Program for Large Network Analysis." *Connections*, **21**(2), 47–57.

Borgatti SP, Everett MG, Freeman LC (2002). ***UCINET*** *for Windows: Software for Social Network Analysis*. Harvard, MA, USA. URL https://sites.google.com/site/ucinetsoftware/home.

Brandes U, Pich C (2007). "Centrality Estimation in Large Networks." *International Journal of Bifurcation and Chaos*, **17**(7), 2303–2318. doi:10.1142/s0218127407018403.

Butts C (2008a). "**network**: A Package for Managing Relational Data in R." *Journal of Statistical Software*, **24**(2), 1–36. doi:10.18637/jss.v024.i02.

Butts CT (2008b). "Social Network Analysis with **sna**." *Journal of Statistical Software*, **24**(6), 1–51. doi:10.18637/jss.v024.i06.

Castro LE, Dong X, Shaikh NI (2018). "Efficient Simulation and Analysis of Mid-Sized Networks." *Computers & Industrial Engineering*, **119**, 273–288. doi:10.1016/j.cie.2018.03.008.

Castro LE, Shaikh NI (2018). "Random Node Pair Sampling-Based Estimation of Average Path Lengths in Networks." *International Journal of Operations Research and Information Systems*, **9**(3), 27–51. doi:10.4018/ijoris.2018070102.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal*, **Complex Systems**(1695).

Dong X, Rangaswamy A, Shaikh NI (2016). "New Methods for Generating Synthetic Equivalents of Real Social Networks." *Available at SSRN 3122037*.

Ebbes P, Huang Z, Rangaswamy A (2016). "Sampling Designs for Recovering Local and Global Characteristics of Social Networks." *International Journal of Research in Marketing*, **33**(3), 578–599. doi:10.1016/j.ijresmar.2015.09.009.

Erdős P, Rényi A (1960). "On the Evolution of Random Graphs." *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, **5**(1), 17–60.

Hagberg AA, Schult DA, Swart PJ (2008). "Exploring Network Structure, Dynamics, and Function Using **NetworkX**." In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15. Pasadena.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Krivitsky PN, Bender-deMoll S, Morris M (2019). ***statnet****: Software Tools for the Statistical Analysis of Network Data*. R package version 2019.6, URL https://CRAN.R-project.org/package=statnet.

Holme P, Kim BJ (2002). "Growing Scale-Free Networks with Tunable Clustering." *Physical Review E*, **65**(2), 026107. doi:10.1103/physreve.65.026107.

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008). "**ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks." *Journal of Statistical Software*, **24**(3), 1–29. doi:10.18637/jss.v024.i03.

Kane M, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(14), 1–19. doi:10.18637/jss.v055.i14.

Koenker R, Ng P (2019). ***SparseM**: Sparse Linear Algebra*. R package version 1.78, URL https://CRAN.R-project.org/package=SparseM.

Leskovec J, Krevl A (2014). "**SNAP** Datasets: Stanford Large Network Dataset Collection." http://snap.stanford.edu/data.

Leskovec J, Sosič R (2016). "**SNAP**: A General-Purpose Network Analysis and Graph-Mining Library." *ACM Transactions on Intelligent Systems and Technology*, **8**(1), 1–20. doi:10.1145/2898361.

Liu G, Zhang M, Yan F (2010). "Large-Scale Social Network Analysis Based on MapReduce." In *2010 International Conference on Computational Aspects of Social Networks*, pp. 487–490. doi:10.1109/cason.2010.115.

Luce RD, Perry AD (1949). "A Method of Matrix Analysis of Group Structure." *Psychometrika*, **14**(2), 95–116. doi:10.1007/bf02289146.

Newman M (2010). *Networks: An Introduction*. Oxford University Press.

Newman MEJ (2002). "The Structure and Function of Networks." *Computer Physics Communications*, **147**(1–2), 40–45. doi:10.1016/s0010-4655(02)00201-1.

Newman MEJ (2003). "The Structure and Function of Complex Networks." *SIAM Review*, **45**(2), 167–256. doi:10.1137/s003614450342480.

Newman MEJ, Strogatz SH, Watts DJ (2001). "Random Graphs with Arbitrary Degree Distributions and Their Applications." *Physical Review E*, **64**(2), 026118. doi:10.1103/physreve.64.026118.

Palmer CR, Siganos G, Faloutsos M, Faloutsos C, Gibbons PB (2001). "The Connectivity and Fault-Tolerance of the Internet Topology." In *Proceedings of the Workshop on Network-Related Data Management (NRDM)*.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Shaikh N, Dong X, Castro L (2020). ***fastnet**: Large-Scale Social Network Analysis*. R package version 1.0.0, URL https://CRAN.R-project.org/package=fastnet.

Uetz P, Dong YA, Zeretzke C, Atzler C, Baiker A, Berger B, Rajagopala SV, Roupelieva M, Rose D, Fossum E, *et al.* (2006). "Herpesviral Protein Networks and Their Interaction with the Human Proteome." *Science*, **311**(5758), 239–242. doi:10.1126/science.1116804.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL https://www.python.org/.

Visser MD, McMahon SM, Merow C, Dixon PM, Record S, Jongejans E (2015). "Speeding Up Ecological and Evolutionary Computations in R: Essentials of High Performance Computing for Biologists." *PLOS Computational Biology*, **11**(3), 1–11. doi:10.1371/journal.pcbi.1004140.

Wasserman S, Faust K (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge. doi:10.1017/cbo9780511815478.

Watts DJ (1999). "Networks, Dynamics, and the Small-World Phenomenon." *American Journal of Sociology*, **105**(2), 493–527. doi:10.1086/210318.

Watts DJ (2003). *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press.

Watts DJ, Strogatz SH (1998). "Collective Dynamics of 'Small-World' Networks." *Nature*, **393**(6684), 440–442. doi:10.1038/30918.

West DB (2001). *Introduction to Graph Theory*. 2nd edition. Prentice Hall, Upper Saddle River.

Weston S (2019). ***doParallel**: Foreach Parallel Adaptor for the **parallel** Package*. R package version 1.0.15, URL https://CRAN.R-project.org/package=doParallel.

Weston S (2020). ***foreach**: Provides Foreach Looping Construct*. R package version 1.4.8, URL https://CRAN.R-project.org/package=foreach.

Yu L, Zheng J, Shen WC, Wu B, Wang B, Qian L, Zhang BR (2012). "BC-PDM: Data Mining, Social Network Analysis and Text Mining System Based on Cloud Computing." In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1496–1499. ACM, New York. doi:10.1145/2339530.2339764.

Affiliation:

Xu Dong
Tamr Inc.
E-mail: dongxu099@gmail.com

Luis Castro
World Bank
E-mail: l.castro6@umiami.edu

Nazrul Shaikh *(corresponding author)*
Cecareus Inc.
E-mail: nazrul@gmail.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2020, Volume 96, Issue 7 *Submitted:* 2017-04-11
doi:10.18637/jss.v096.i07 *Accepted:* 2018-10-29