Complex Network Analysis: The Need for Speed

WANDELT Sebastian[1] and SUN Xiaoqian[1*]

1. National Key Laboratory of CNS/ATM, School of Electronic and Information Engineering, Beihang University, Beijing 100191, P. R. China

E-mail: wandelt@informatik.hu-berlin.de, sunxq@buaa.edu.cn

Abstract: Complex network analysis has become a critical research area for the design and evaluation of complex systems, due to the increasing success of modeled phenomena. Accordingly, there is a growing need for easy-to-use and fast network analysis software packages. In this work, we report on a comprehensive benchmark of state-of-the-art network analysis software. We perform exhaustive experiments on a set of eight competitors for 15 important network analysis tasks. All methods are evaluated on 13 publicly available datasets. We find that none of the competitors clearly dominates our competitive evaluation, while for specific tasks, the running times differ by two orders of magnitude and more. Our study helps researchers to select an appropriate tool for their network analysis tasks.

Key Words: Complex Networks; Network Analysis Software; Benchmark

1 Introduction

During the last decades, researchers have successfully modeled many phenomena and systems as complex networks, which, essentially, consist of nodes interrelated by links. In general, complex network (models) can be applied to a multitude of different domains, they have in common that, in the end, solely from the structure of relationships between entities, one can draw insights into complex systems. According to the growing prevalence of networks, the number of complex network analysis techniques is rising rapidly. At the same time, the size of networks is increasing at an unpreceded speed, which makes network analysis a challenging task. Network scientists, in practice, often need to perform exploratory analysis on network data. For this purpose, different network analysis software have been developed, often as part of open source/open data initiatives, which help researches to perform their complex network experiments. Over the last years, the landscape of analysis tools has become rather heterogeneous, with many competing systems.

Therefore, before performing actual experiments, the problem is to select an appropriate software, which is often a difficult undertaking. In this work, we report on the (to the best of our knowledge) most comprehensive benchmark on state-of-the-art network analysis software. We perform our experiments on a set of eight competitors, implemented in multiple high-level programming languages (Python, R, and Java): *Gephi*, *graph-tool*, *JUNG*, *NetworkX*, *NetworKit*, *Snap.py*, *igraph-python*, and *igraph-R*[1]. These tools have been frequently used in recent research papers, and therefore, have a proven track record as relevance for science. The eight competitors are evaluated on 13 distinct datasets, all of which have been used in research papers as well. The datasets are available for free download from the UCI Network data repository[2]. These datasets include: *karate*, *dolphins*, *lesmis*, *polbooks*, *adjnoun*, *football*, *celegansneural*, *polblogs*, *netscience*, *power*, *hep-th*, *astro-ph*, and *condmat*. For our study, we choose a collection of tasks, ranging over a wide set of applied network problems. This includes, data management tasks (such as *loading networks into main memory*, *iterating over nodes*), the computation of eight centrality measures (such as *betweenness* and *closeness*), and additional tasks (such as *shortest path computation* and *modularity detection*).

The wealth of experiments we have performed with state-of-the-art network analysis tools and the number of programs we compared allow us to draw several interesting conclusions about scalability of these tools for large datasets. Moreover, given that the competitors and data are available for free download, our experiments can be repeated and the results can be reproduced by others.

The major contributions of this paper are: 1) We report on the **scalability of network analysis software**. 2) We compare 8 **state-of-the-art competitors** on 13 datasets for 15 competition tasks. 3) Our results show **wide range of running times**. We find that given a specific task, the running times differ up to more than **two orders of magnitude**. While there is no single winner for all tasks, the results of our study can help researchers to select an **appropriate tool for performing their experiments**.

The remainder of this paper is structured as follows. We describe the preliminaries for our experiments in Section 2, including the evaluation tasks, competitors and datasets. Section 3 presents the results of our experimental evaluation of standard network analysis tools against the tasks defined in Section 2. The paper is concluded in Section 4.

2 Preliminaries

This section introduces the preliminaries for our experimental study on network analysis software. In Section 2.1, we introduce a set of tasks, which we used for evaluating state-of-the-art network analysis software packages. The competing software packages are introduced in Section 2.2. We evaluate the competitors on a set of standard datasets with long traditions in research; these datasets are introduced in Section 2.3. Section 2.4 describes the setup of our comparative evaluation.

2.1 Tasks for complex network analysis

We distinguish three evaluation tasks: Network-management related tasks, node centrality computation, and additional operations. These tasks are introduced below.

[1]The authors of this study are not affiliated with any of these eight software systems, which ensures an unbiased evaluation as any user will experience it.

[2]https://networkdata.ics.uci.edu/index.php

Network-management related tasks The first step for analyzing a complex network with a software package is to load the file from hard disk into main memory. While this a singular event, it is still interesting to compare the running times for this process. Furthermore, recurring tasks in network analysis include the traversal of all nodes/links and the time for removing nodes, for instance, during the simulation of an attack to the network. Furthermore, it is interesting to compare the amount of main memory used, since high processing speeds often come at the price of precomputed data structures, which need to be kept in main memory. For large graphs, the free main memory is often a realistic constraint. Overall, we investigate the following five data management-related measures for all competitors:

1) *Loading time*
2) *Time for traversing all nodes*
3) *Time for traversing all links*
4) *Time for removing nodes form the network*
5) *Main memory consumption*

Centrality score computation In complex networks, it is often desired to identify the importance of a node for the network. This can be characterized regarding several roles/criteria. For this purpose, researchers have defined a large number of centrality measures, each of which is tailored specifically towards the notion of importance at hand. Below, we revisit eight centrality metrics, which have been widely used in academic studies and are mostly implemented in our eight chosen network analysis software packages:

1) *Betweenness centrality:* $B_i = \sum_{s \neq t} \frac{\sigma_{st}(i)}{\sigma_{st}}$, where σ_{st} is the number of shortest paths going from node s to node t; $\sigma_{st}(i)$ is the number of shortest paths going from node s to node t and passing through node i [1]. This metric indicates the number of shortest paths going through a node.
2) *Closeness centrality:* $C_i = \frac{\sum_{j \in N, j \neq i} \sigma_{ij}}{(n-1)}$, where N is the set of all nodes in the network, n is the number of nodes, σ_{ij} is the shortest path between node i and node j. This metric is the average distance from a given starting node to all other nodes in the network [1].
3) *Clustering coefficient:* $CC_i = \frac{\sum_{j,k} a_{ij} a_{ik} a_{jk}}{k_i(k_i-1)}$, where a_{ik} is the connection between node i and node k, a_{jk} is the connection between node j and node k. This metric gives an overall indication of how nodes are embedded in their neighborhood.
4) *Degree centrality:* $D_i = \sum_j a_{ij}$, where a_{ij} is the connection between node i and node j: $a_{ij} = 1$ when there is a connection existing; $a_{ij} = 0$ otherwise. For all pairs of nodes in the network, a_{ij} composes the adjacency matrix. Degree refers to the number of connections with other nodes.
5) *Eigenvector centrality:* This metric measures the influence of a node in the network. It is the eigenvector for the largest eigenvalue of the adjacency matrix. Nodes with high eigenvector centrality also connect to other nodes which have high eigenvector centrality.
6) *HITS:* HITS Authorities measures how valuable information stored in one node is, based on incoming links, while HITS hubs estimates how valuable information stored in one node is, based on links [2].
7) *Katz centrality:* A centrality measure extending eigenvector centrality, taking into account the importance of its direct neighbors. Opposed to other centrality measures, it does not only consider shortest paths, but also the total number of walks.
8) *Page rank:* A ranking of the nodes based on the structure of the incoming links. The intuition is that important nodes have a higher probability to receive many incoming links.

Additional network analysis tasks

We investigate two additional tasks, which are quite common in the analysis of network robustness. The first task is to *compute the shortest paths* between all node pairs in the network. Common applications include transportation (vehicle routing), telecommunication (packet routing), social networks, and VLSI design. The second task addresses the problem of *computing a community structure* in a network. For our experiments, we compute communities using the Louvain method [3], which is based on greedy optimization.

2.2 Competitors

During the last decade(s), the increased interest in network analysis, particularly spurred by emerging social networks, has led to a large number of network analysis tools, many of which are freely available as part of the Open Source Initiative. While the number of truly general software packages (with a release state beyond alpha) is rather limited, there exists thousands of smaller specific packages. The latter ones are disregarded for our study, since our focus is on general network analysis. We have selected a set of eight competitors, all of which have been proven successful with a track record in scientific publications. Our competitors cover a wide range of implementation focus and also different programming languages: Java, Python, and R. We have intentionally chosen these high-level languages, since many (non-computer) scientists prefer data-centered languages over more traditional system-oriented languages, such as C++. In total, we compare the following eight competitors in our study:

1) *GEPHI*: Gephi [4], implemented in Java, can be considered one of the first network analysis software packages. Its development had stopped in the beginning of 2013, but a new release has just been published in December 2015. We have used Gephi 0.9.0[3] in our study.
2) *NETWORKX*: Based on Python, NetworkX [5] is undoubtedly one of the most used network analysis packages, mainly due to its simple installation and intuitive use out of the box. We use NetworkX 1.9.1[4] in our study.
3) *NETWORKIT*: NetworKit [6], based on python, is a recently published tool for high-performance network analytics, with focus on approximations and multi-core implementations. We use NetworKit 4.0.1[5] in our study.

[3] https://gephi.org/
[4] https://networkx.github.io/
[5] https://networkit.iti.kit.edu/

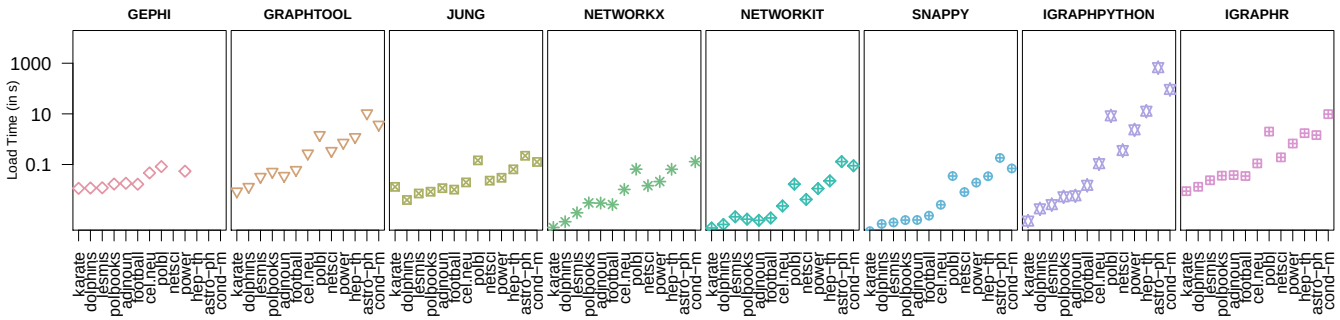

Fig. 1: Time for loading datasets from hard disk into main memory, reported in seconds. The load time increases significantly with the number of nodes/links in the dataset. The differences between competitors are up to two orders of magnitude. IGRAPHPYTHON has the longest load time (more than 11 minutes), while NETWORKIT and SNAPPY can load even the largest dataset within a few seconds.

4) *JUNG*: The Java-based JUNG [7] is a framework for modeling, analysis and visualization of data. We have used JUNG 2.0.1-3[6] for our experiments.
5) *SNAPPY*: A Python interface for SNAP, which is written in C++ and optimized for compact graph representation and high performance. We used Snap.py 1.2.2.4[7] for our experiments.
6) *IGRAPHR:* The network analysis toolset igraph is built with emphasis on efficiency and portability. igraph-R is the python-binding of igraph. We have used igraph-R 1.0.1[8] for our experiments.
7) *IGRAPHPYTHON:* The python-binding of igraph. We have used version igraph-python 0.1.11[9] for our study.
8) *GRAPHTOOL*: The core of the network analysis software graph-tool is implemented in C++ and based on the Boost Graph Library; The authors also provide a Python interface for graph-tool. We used this interface graph-tool 2.12[10] in our study.

2.3 Datasets

Our experiments were conducted with 13 datasets from the UCI Network data repository[8], which is proved by the UCI Datalab for increasing the study of networks, as part of the Open Data Initiative [9]. All these networks have been used frequently in scientific publications on network analysis. The datasets cover a wide range of application domains, such as biology, politics, and research. A summary of the topological features is provided in Table 1. In our study, we regard all networks as undirected, since several network problems are only defined for undirected networks. Whenever there are links between two nodes in both directions, we merge them into an undirected link. In general, most of these networks are rather sparsely connected, with density values (number of links divided by the maximum number of links) less than one percent. Furthermore, the networks often have small-world / scale-free properties.

The datasets *karate*, *dolphins*, *lesmis*, *polbooks*, *adjnoun*, *football*, and *celegans.neural* are rather small, with less than 1000 nodes and less than 10,000 links. The majority of these networks can be easily handled by all competitors in our study. The datasets *polblogs*, *netscience*, *power*, *hep-th*. *astro-ph*, and *cond-mat* are larger and pose challenges to many of the existing systems for network analysis, i.e. solving standard network problem often cannot be performed in a few seconds. It should be noted that the network *power* has an outstandingly high average shortest path length, while *netscience* has the highest clustering coefficient (a measure of the degree to which nodes in a network cluster together). In Table 1 and also in the evaluation section, we have used abbreviated names for these datasets. All datasets are available for free download, which allows to reproduce our experiments.

[6] http://jung.sourceforge.net/
[7] https://snap.stanford.edu/snappy
[8] http://igraph.org/r/
[9] http://igraph.org/python/
[10] https://graph-tool.skewed.de/

Table 1: Topological properties of the datasets used in our study. ASPL=Average Shortest Path Length, CC= Clustering Coefficient. Maximum values are highlighted in bold. ASPL and CC have been computed based on the giant component of the networks.

Dataset	Nodes	Links	Density	ASPL	CC
karate	34	78	**0.011**	2.40	0.57
dolphins	62	159	0.004	3.35	0.25
lesmis	77	254	0.0023	2.64	0.5
polbooks	105	441	0.001	3.07	0.48
adjnoun	112	425	0.0012	2.53	0.17
football	115	613	0.0006	2.50	0.40
cel.neu.	297	2148	0.0001	2.45	0.29
polbl	1224	16715	8.7E-006	2.73	0.32
netsci	1461	2742	0.0003-	6.04	**0.74**
power	4941	6594	0.0002	**18.98**	0.08
hep-th	7610	15751	6.1E-005	7.02	0.50
astro-ph	16046	**121251**	2.1E-006	4.79	0.66
cond-m	**16264**	47594	1.4E-005	6.62	0.65

2.4 Setup

The experiments in our study were performed on a Dell PowerEdge T630 with 2* Intel Xeon E5-2640 v3 @ 2.6 GHz with 224 GB RAM. In total, the system provides 32 cores. The operation system was Fedora 21 with Linux Kernel 4.1.13-100.fc21.x86_64. All competitors were installed following their default setup procedures. We did not tweak any settings, such as amount of available main memory or allowed number of cores, but rather tested out-of-the-box configurations.

3 Results

We report our results below. First, we analyze the running times for loading, iterating, and manipulating network data

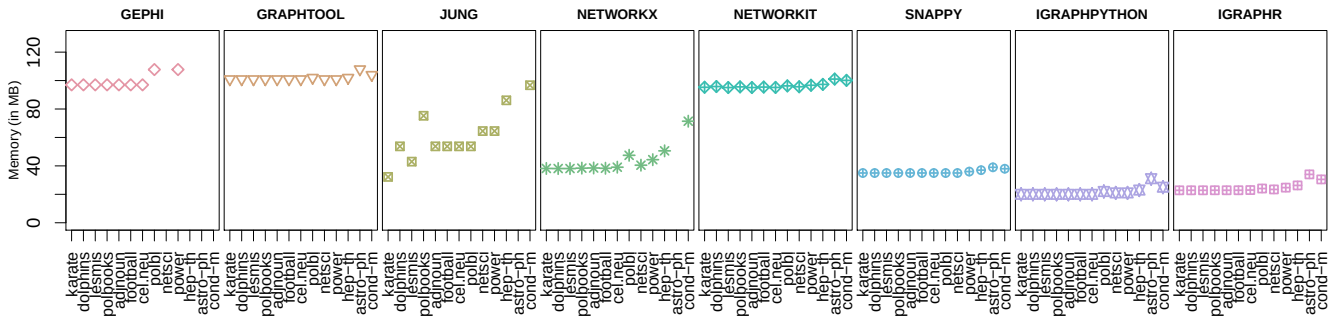

Fig. 2: Amount of main memory in MB (as reported in max RSS) for each competitor and dataset. The competitors have very distinct baselines: IGRAPHPYTHON only requires 20 MB for all its standard code and data structures, while GRAPHTOOL occupies 90 MB even for the smallest dataset in our study. Furthermore, the raise in memory usage with an increasing number of nodes/links varies a lot between competitors.

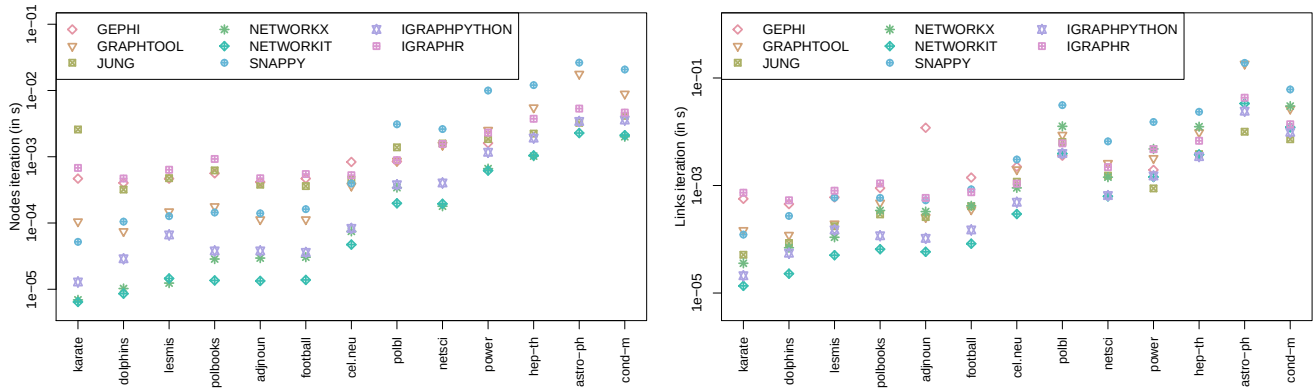

Fig. 3: Time in seconds required for traversing all nodes (left) and links (right) in the network. Even for larger graphs, nodes and links can be iterated within a few hundred milliseconds. Yet, the differences in running time are in orders of magnitude again.

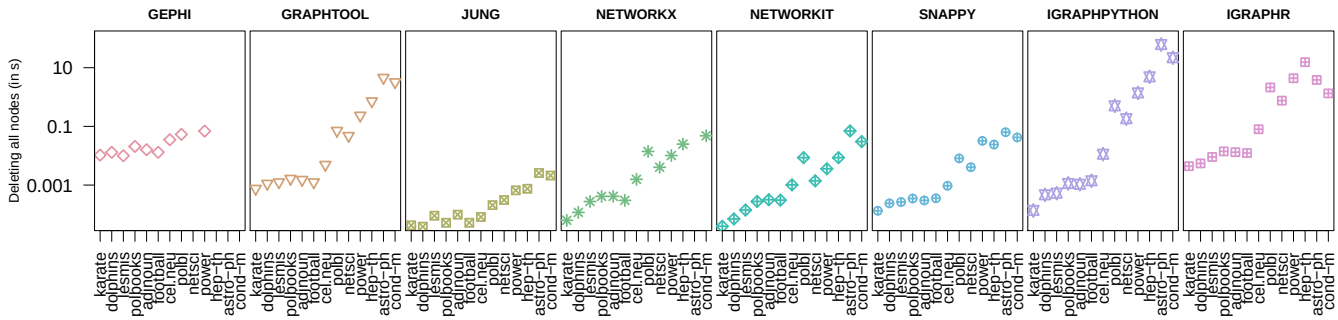

Fig. 4: Time in seconds required for deleting all nodes in the network. Nodes have been removed in a random order from the network; all times were averaged over 50 random node orderings. GRAPHTOOL and IGRAPHPYTHON are the slowest competitors here, while JUNG is up to two orders of magnitude faster.

structures inside the competitors in Section 3.1. Second, we report the running times for computing eight centrality measures in Section 3.2. Additional tasks are evaluated in Section 3.3.

3.1 Network-management related tasks

First, we report the load time for all datasets in Figure 1. For all competitors, we have loaded the network data from a csv-file, which contained first all nodes and then all links. The parsing of the file for each programming language was implemented by the authors; Afterwards, we used competitors' interfaces to add single nodes and links. Therefore, the loading time can be fairly compared between all competitors. Yet, it can be seen that the load time is significantly different between competitors: IGRAPHPYTHON has the longest load time (more than 11 minutes), while NETWORKIT and SNAPPY can load even the largest dataset within a few seconds. In general, longer load times are often caused by computing additional index structures on-the-fly, for instance, as it can be seen in GRAPHTOOL. This effect is reflected in the amount of main memory used, as shown in Figure 2. Graphtool has the highest main memory usage among all competiors, while SNAPPY, which was rather fast in loading, has a rather low memory footprint. Overall, we find that some of the competitors scale rather well with the increasing number of nodes in the graph (albeit having a rather high baseline), while other' main memory usage is significantly increased, for instance, JUNG and NETWORKX.

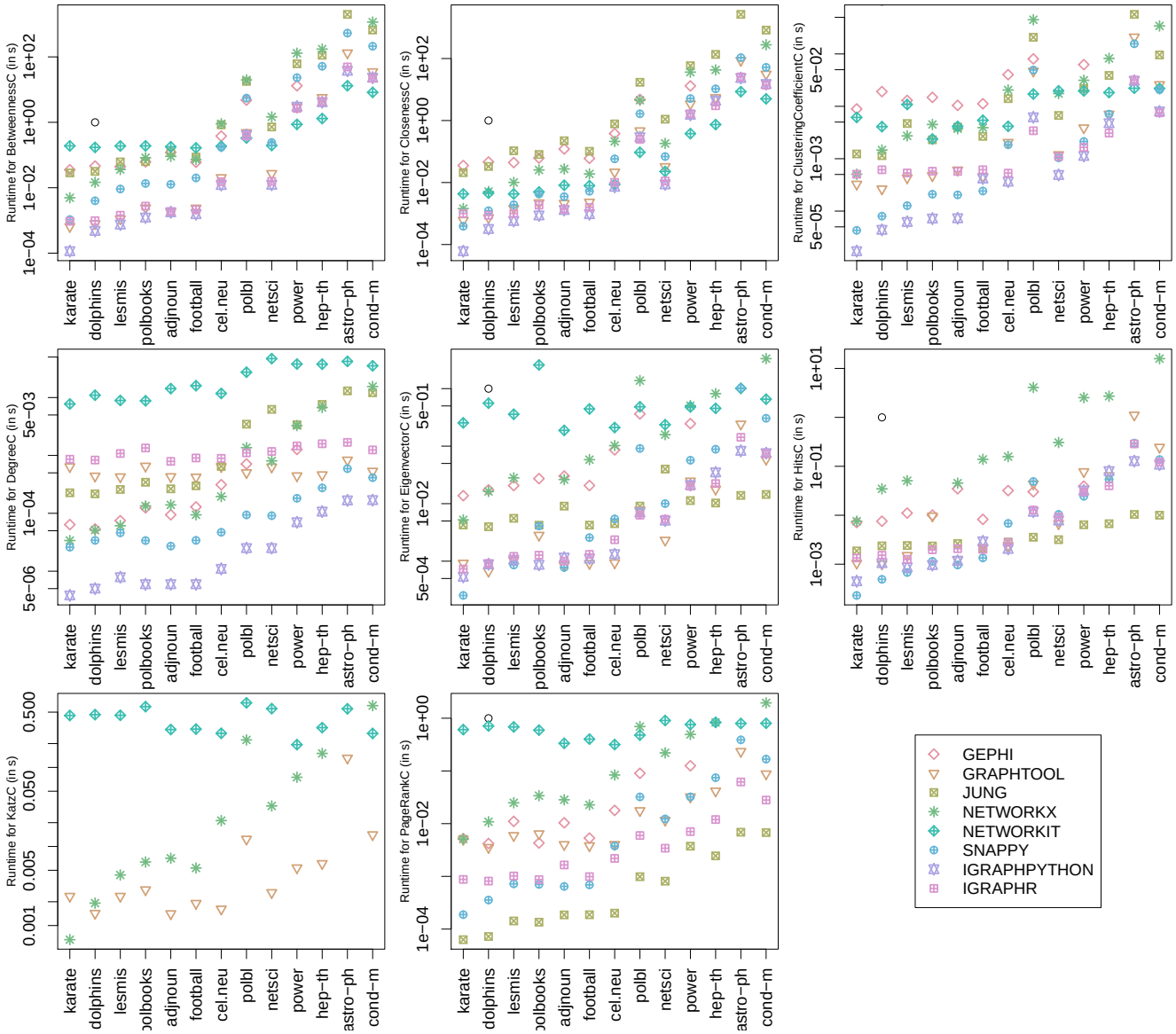

Fig. 5: Running time for computing centrality measures for the datasets. From the upper left: *betweenness*, *closeness*, *clustering coefficient*, *degree*, *eigenvector*, *HITS*, *Katz*, and *PageRank*. Running times for the same measure vary by orders of magnitude. However, there is no single competitor dominating all centrality computations.

The time for traversing nodes and links in a network is reported in Figure 3. Iteration over all elements is rather fast (within a few hundred milliseconds) for all competitors. Only if an iteration is at the heart of an algorithm implementation, it makes sense to select a competitor which guarantees– fast iteration, such as, NETWORKIT. In Figure 4, we show the time measured for deleting all nodes from a network. Again note that running times vary by up to two orders of magnitude. The slowest competitors are GRAPHTOOL, IGRAPHPYTHON, and IGRAPHR. We think that updating the graph requires more time for approaches that use indexing on top of the pure graph structure, where frequent updates to the network also induce (expensive) updates to the index.

3.2 Centralities

In Figure 5, running times for the computation of the eight centrality measures are shown. The rankings among competitors are rather different, yielding a heterogeneous results for that part of our study. NETWORKIT is fastest for computing *betweenness* and *closeness* centrality. IGRAPHPYTHON is ranked top for computation of *clustering coefficient* and *degree*. JUNG is the fastest competitor for *eigenvector*, *HITS*, and *Page rank*. Finally, GRAPHTOOL is best suited for efficient computation of *Katz* centrality. Overall, we can observe that NETWORKIT is rather slow for small graphs, but becomes more efficient for computing centralities of larger graphs; This is a recurring pattern for all centralities.

3.3 Others

First, we have analyzed the running times for solving the *all pairs shortest path* problem; the results are shown in Figure 6(left). The fastest competitors are GRAPHTOOL, IGRAPHR, and IGRAPHPYTHON. GEPHI, on the other hand, could not even finish the computation for the largest graphs within one hour. Second, we report the running times for modularity detection with the Louvain method in Fig-

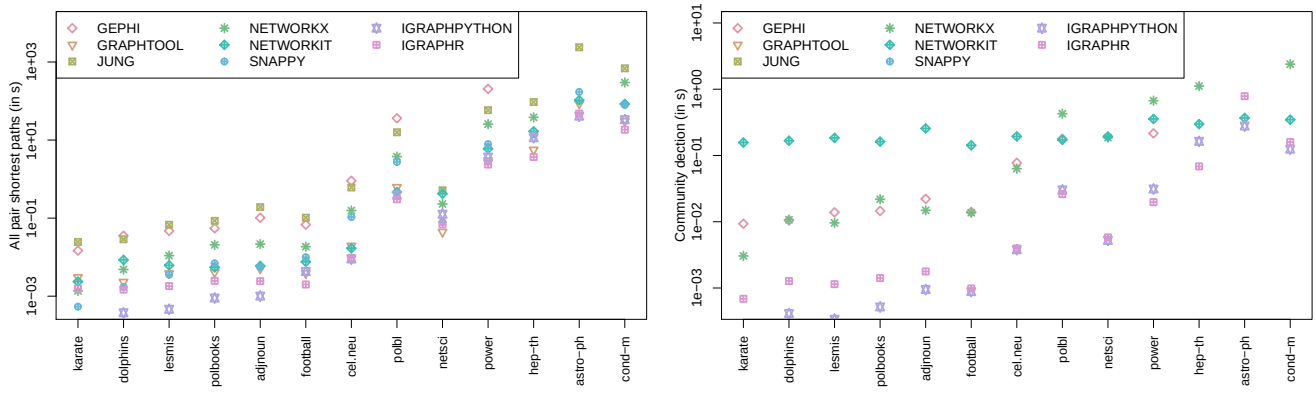

Fig. 6: Time in seconds required for computing all pairs shortest paths (left) and modularity using the Louvain method (right). For both problems IGRAPHR and IGRAPHPYTHON are the fastest competitors.

Table 2: Summary of the evaluation results for our study. We present the average ranking of the competitor for each task over all datasets. Minimum values (=better) are indicated in bold. The summary line at the end of the table counts how often a competitor is ranked best.

Problems	GEPHI	GRAPHTOOL	IGRAPHPY	IGRAPHR	JUNG	NETWORKIT	NETWORKX	SNAPPY
BetweennessC	6.385	2.923	**1.692**	2.538	6.462	4.769	6.615	4.615
ClosenessC	7.308	3.462	**1.615**	2.462	7.462	3.154	6.154	4.385
Clustering Coeff.	7.769	3.769	**1.385**	2.769	5.769	5.462	6.462	2.615
Community Det.	4.000	5.923	**1.231**	1.846	6.923	4.154	3.923	8.000
DegreeC	5.000	4.538	**1.000**	5.846	5.538	7.615	4.462	2.000
Delete	7.385	5.615	5.846	6.846	**1.077**	2.308	3.846	3.077
EdgesIter	6.692	5.000	2.154	5.846	3.308	**1.538**	4.769	6.692
EigenvectorC	6.769	**2.308**	2.538	2.769	3.615	7.154	7.000	3.846
HitsC	6.000	3.846	**2.615**	3.154	2.846	8.000	6.769	2.769
KatzC	3.923	**1.077**	7.000	8.000	4.923	2.846	2.231	6.000
LoadTime	6.308	6.538	5.692	6.462	4.462	**1.308**	3.538	1.692
Memory	7.385	7.385	**1.000**	2.000	4.923	5.846	4.385	3.077
NodesIter	6.615	4.923	3.000	6.538	5.692	**1.308**	2.154	5.769
PageRank	5.538	3.615	8.000	2.462	**1.000**	6.462	5.923	3.000
Shortest Paths	7.538	3.077	1.923	**1.769**	7.308	4.538	5.769	4.077
|Best ranking|:	0	2	7	1	2	3	0	0

ure 6(right). Again, IGRAPHPYTHON and IGRAPHR are the fastest competitors for all datasets.

4 Conclusions

In this paper, we have compared eight state-of-the-art network analysis tools for 13 datasets on 15 common analysis tasks. Interestingly, none of the competitors shows a dominance, but tools should be carefully selected based on the task at hand. We summarize the results of our study in Table 2. IGRAPHPYTHON is ranked best (on average over all datasets) seven out of 15 times, closely followed by NETWORKIT, which is ranked best three times. For future work, it would be interesting to evaluate the top-ranked competitors on even larger datasets and more analysis tasks.

References

[1] L. C. Freeman, "Centrality in social networks: Conceptual clarification," *Social Networks*, vol. 1, pp. 215–239, 1978.

[2] J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008 (12pp), 2008. [Online]. Available: http://stacks.iop.org/1742-5468/2008/P10008

[4] M. Bastian, S. Heymann, M. Jacomy *et al.*, "Gephi: an open source software for exploring and manipulating networks." *ICWSM*, vol. 8, pp. 361–362, 2009.

[5] D. A. Schult and P. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, 2008, pp. 11–16.

[6] C. Staudt, A. Sazonovs, and H. Meyerhenke, "Networkit: An interactive tool suite for high-performance network analysis," *CoRR*, vol. abs/1403.3005, 2014. [Online]. Available: http://arxiv.org/abs/1403.3005

[7] J. OMadadhain, D. Fisher, S. White, and Y. Boey, "The jung (java universal network/graph) framework," *University of California, Irvine, California*, 2003.

[8] C. L. DuBois, "UCI network data repository," 2008. [Online]. Available: http://networkdata.ics.uci.edu

[9] D. Partha and P. A. David, "Toward a new economics of science," *Research policy*, vol. 23, no. 5, pp. 487–521, 1994.