Project Report

Dissecting The Complex: Investigating the potential of shared memory vertex-centric programming for network analysis and the potential of network-based node importance algorithms for ranking the importance of nodes in social networks

Theresa Hradilak
theresa.hradilak@student.hpi.de

Robin Wersich
robin.wersich@student.hpi.de

Hasso-Plattner-Institute, University of Potsdam
Working Group Data Analytics and Computational Statistics

1 Abstract

Even with increasing computational power, processing large networks ($> 10^6$ nodes) quickly reaches its limits on standard node clusters, e.g. when calculating node centrality measures, whose algorithms often have a quadratic run-time complexity. To improve network processing capability, vertex-centric programming (VCP) targets creating a system for large-scale graph processing which is as efficiently parallelizable as programmable. While vertex-centric programming was originally proposed for distributed memory computing, recent publications also suggest beneficial usage on shared memory systems.
We investigated the potential of shared-memory vertex-centric programming as implemented by iPregel regarding its performance and programmability. We compared run-time, memory usage and impact of parallelization for Single Source Shortest Path (SSSP), PageRank and Connected Components algorithms between iPregel and the non-vertex-centric graph processing library NetworKit. To evaluate the programmability provided by iPregel, we tested implementing Betweenness and Closeness Centrality in iPregel and evaluated iPregels implementation of the proposed vertex-centric programming API by Pregel.
We found that iPregel shows a higher parallelization power than NetworKit when comparing algorithms parallelized in both frameworks. iPregel can parallelize algorithms where NetworKit does not work with parallelization (Connected Components, SSSP). The performance varies depending on the tested algorithm. For example, while iPregel showed up to 10 times faster runtimes for PageRank, it was up to 10 times slower in calculating Connected Components. In regards to programmability, due to design choices made for iPregel, we were not able to implement Betweenness and Closeness Centrality. In general, we found that while iPregel proves that potential for performance improvement using shared-memory vertex-centric programming exists, iPregel itself is currently not suited as a general-purpose graph analysis framework.

As an application of the investigated graph processing frameworks, we employed the selected node centrality graph analyses on social networks. Traditionally, easy-to-calculate measures of importance are used to estimate the relevance of persons in real-world networks, e.g., the h-index for scientists or the follower count for influencers. However, with the continuing increase in computational power, other more computationally intensive measures based on network algorithms like PageRank, Betweenness Centrality, and Closeness Centrality become practicable.
We analysed the feasibility of PageRank, Betweenness Centrality, and Closeness Centrality regarding their fitness as general importance measures and the relationship between classical node importance measures and the newly proposed measures on four real-world networks such as a citation network and a Twitter interaction network.
We found that all three investigated importance measures (PageRank, Betweenness Centrality, and Closeness Centrality) are feasible importance measures on the tested networks regarding their capability of distinguishing nodes in a network. Regarding their relationship with classical measures, we observed a strong correlation between the h-index and all three investigated importance measures for the cit-HepTh citation network and, in contrast, no correlation with the follower count of users in the higgs-Twitter interaction network.

2 Motivation and background

2.1 Shared Memory Vertex-Centric Programming & Performance

Networks can be used to model many domains and problems of interest, from naturally emerging applications like road and social networks to other complex systems like the human body. In recent years network analysis has been used extensively for research, with 380.000 papers registered with the keyword network analysis by Google Scholar in 2020 alone. While network analysis can deliver new insights, many algorithms for network measures have a high runtime complexity. In this years' Dissecting The Complex seminar, employed graph analyses with at least quadratic runtime algorithms were, e.g. Betweenness Centrality [Brandes, 2001], and Closeness Centrality [Okamoto et al., 2008] for network centrality measures, as well as Walktrap [Pons and Latapy,] for community detection.
Combined with the increasing size of available networks of interest, the described high runtime complexity can lead to weeks or more of runtime on common node clusters available at universities. These challenges became apparent during the Dissecting The Complex seminar when multiple groups had to stop calculations on the scorelab cluster because of runtime limitations. As demonstrated by this example, the described challenges regarding the runtime of network analysis algorithms pose limitations to possible research projects and seminars.
Besides finding algorithms with better runtime complexity, which will not be possible for many problems with provable quadratic lower bounds [Kintali, 2008], or getting access to systems with higher computational power, a third option would be parallelizing calculations. However, network analysis algorithms often suffer from poor parallelizability [Malewicz et al., 2010]. Parts of the poor parallelizability of classical graph algorithms are based on their poor locality of memory access, meaning that consistently different storage areas need to be accessed, which blocks efficient usage of caching. Additionally, graph algorithms often exhibit "little work per vertex, and a changing degree of parallelism over the course of execution" [Malewicz et al., 2010]. These issues are only aggravated when scaling to distributed systems.
Pregel [Malewicz et al., 2010], introduced in 2010, aims to address the challenges in large-scale graph processing with the concept of vertex-centric programming. In vertex-centric programming, each computation consists of a series of supersteps. In each superstep each vertex performs a user-defined function on its vertex-specific data and can pass new data to other vertices (mostly its neighbors) via messages. Pregel promises efficient parallelization and scaling capability on distributed systems combined with a high level of programmability.
While distributed systems offer higher computational power and can be the only way of processing huge graphs like the web graph, they are also more challenging to set up, maintain and run than non-distributed systems. This overhead makes them less accessible for scientists at universities and motivates the question whether profitable usage of vertex-centric programming on shared memory systems is also possible, as newer publications already suggest [Capelli et al., 2019a].
Due to the introduced overhead of message-based communication in vertex-centric programming, which facilitates distributing systems but appears unnecessary on shared-memory systems, we were interested in the impacts on runtime and parallelization on shared-memory systems. Specifically, we investigated the shared-memory vertex-centric programming framework iPregel and compared its parallelization power, runtime and memory efficiency for selected networks and network analysis algorithms against NetworKit.

2.2 iPregel & Programmability

The second significant aspect of Pregel is the proclaimed high programmability of its API, allowing users to express generic algorithms within a simple interface. Pregels API also abstracts from its parallelization implementation so that users don't need to worry about parallelization when implementing their algorithm. The ease of use for writing new well-parallelizable graph processing algorithms is essential, as the other graph processing libraries we evaluated often did not allow much modification. Usually, to adapt algorithms provided by graph processing libraries, one needs to adapt the source code itself, dealing with sometimes complex parallelization code. If researchers need to adapt algorithms for their projects, they could benefit from the user-friendly API offered by Pregel. Therefore we were interested in the programmability of vertex-centric programming and which kind of algorithms are well expressable with it.

2.3 Alternative Node Importance Measures Analysis

Measures of influence and importance in human structures impact our daily life. For example, grants for scientists or endorsement deals for influencers often depend on measures taking into account only a few variables, e.g. the follower count or a combination of paper quantity and citation count. The idea of assigning single numbers to persons and rating their importance based on these numbers is, of course, leaving important factors out at best and actively

harmful at worst. Nevertheless, as much of our lives depend on such measurements, we should work towards measures that are as fair as possible. And as life is multifaceted, but many of the currently employed measures only factor in few variables, one way to improve importance measures could be by taking into account more complex information on the social networks for whose members importance measures should be calculated. More balanced importance measures could make decisions based on the calculated scores fairer and, e.g. award research grant money in a more just manner.
Improving importance measures by basing them on network analysis is not a new thought as research into alternatives for the h-index in the form of a Pagerank-Index [Senanayake et al., 2015] or PR-Index [Gao et al., 2016] shows.
As a first step in exploring potential alternative node importance measures for social networks, it is relevant to identify network-based measures that could serve as importance measures. In this project, we evaluated three well-known node centrality measures PageRank, Betweenness Centrality, and Closeness Centrality, regarding their ability to distinguish nodes and their relationship to classically used measures.

3 Datasets

3.1 Source Data

As we wanted to compare node importance scores generated by graph analysis with classical ones, we looked for networks with nodes that ideally already had some sort of classical node importance measure. Datasets of interactions between humans proved to be a good choice for that as people can be and are rated in many different ways and the results are of great importance. We settled with the four networks shown in Table 1 which we obtained from the SNAP [Leskovec and Krevl, 2014] and ICON [Aaron Clauset and Sainz, 2016] network databases:

Network Dataset	**Nodes**	**Edges (Arcs)**	**Type**	**Source**	**Classical measure of importance**
Arxiv High Energy Physics author citation network	8,731	461,108	directed	generated from *cit-HepTh*[1], SNAP	h-index
Network of Twitter interactions (replies, retweets, mentions)	456,631	461,192	directed	generated from *higgs-Twitter*[2], SNAP	follower count
Network of stackoverflow interactions (answers, comments)	2,601,977	34,875,684	directed	generated from *sx-stackoverflow*[3], SNAP	reputation count (could not be retrieved)
IMDb Actor collaboration network	2,180,759	228,985,632	undirected	*hollywood-2011*[4], ICON	awards, famousness

[1] `https://snap.stanford.edu/data/cit-HepTh.html`
[2] `https://snap.stanford.edu/data/higgs-Twitter.html`
[3] `https://snap.stanford.edu/data/sx-stackoverflow.html`
[4] `http://law.di.unimi.it/webdata/hollywood-2011/`

Table 1: The networks we chose for analysis and benchmarking.

3.2 Data Preprocessing

None of our networks could be used in the way we intended out of the box, so we needed to perform individual preprocessing on each:

Citation Our source was a network of papers with directed edges indicating citations. In addition, SNAP provided a metadata text file for each paper, including the authors. We extracted these for each paper and used this information to generate a graph of authors where a directed edge from A to B was added if at least one paper authored by B was cited by at least one paper authored by A. Self references of authors were removed. In

addition, we generated a metadata file with author name and h-index [1] for each author, where the latter was calculated using the number of papers of each author and their respective citation count.

A significant problem when extracting the authors for each paper was that the paper metadata was not standardized. Authors were separated by varying combinations of commas, 'and's and spaces and often included their research institute in parentheses. Sometimes, the page count of the paper was added at the end of the author list. In addition, different abbreviations were used for the same author, and spelling mistakes were present. We decided to identify each author by their full last name and first letter of the first name, which appeared to be a good enough merging strategy. As we calculated the h-index on the same data basis, possible errors in this approach should have no major impact on possible correlations later on.

Twitter The provided Twitter network contained an edge from A to B if user A answered or retweeted a tweet by B or if A mentioned B in a tweet. Each edge was labeled with a timestamp and the concrete interaction type. We removed both in order to treat each interaction equally. In addition, the resulting duplicates, as well as self-references, were removed. To obtain the follower count for each user, we used the follower graph provided by SNAP, which contained all users involved in the above-described interactions and calculated the in-degree.

stackoverflow Similar to the Twitter network, the stackoverflow network contained an edge from a user A to a user B if A answered or commented a question (or answer) of user B, labelled with a timestamp which we removed. Again resulting duplicates, as well as self-references, were removed. Unlike on Twitter, on stackoverflow not users that are *interacted with* (high in-degree) are important, but users that *interact* (high out-degree), because they are the people *knowledgeable* enough to answer questions. Consistently, people who can answer questions of people with great knowledge must have great knowledge themselves (similar to how people mentioned by influential people on Twitter also gain influence). For this reason, we decided to reverse the edges in the stackoverflow graph, so people with high knowledge also have a high in-degree and will thus be considered important by algorithms like PageRank.

We originally intended to extract the reputation count from the dump from which the graph was created. Sadly, SNAP only references the website from which the dump for various years can be downloaded. It does, however, not state which exact dump was used to generate the graph in question, so we were not able to retrieve a reputation count for the stackoverflow users.

IMDb The IMDb actors (2011) dataset contains an edge between two actors A and B if they were credited in the same movie of the IMDb movie database. Unlike the others, this dataset was not obtained from SNAP and came in the compressed binary WebGraph[2] format. We created a small Maven project based on a *WebGraphDecoder*[3] from GitHub Gist to convert the graph to the SNAP edge list format. We additionally downloaded the provided metadata file mapping actor IDs to names and the Academy Award Nominee List from Wikipedia as possible references for evaluating the node importance scores.

In addition to the dataset individual preprocessing, we performed some conversions on all of our four networks:

1. In our directed networks, important nodes have a high in-degree because many other nodes reference them. This design makes sense for PageRank but not for distance-based importance measures, like Betweenness and Closeness Centrality. For those, we reversed the graphs (so that important nodes have many out-edges, representing influence) to properly calculate the distance to other nodes along the influence flow.

2. The Connected Components algorithm provided by iPregel only works on undirected graphs. Therefore we decided to create an undirected version of each graph for this algorithm (that is, every edge is present in both directions as iPregel is incapable of doing the mirroring by itself). The obtained results are the same as executing Weakly Connected Components on the original, directed graphs.

3. While NetworKit supports the SNAP edge list format, iPregel expects a binary input format. It is claimed that the input must be in the binary format used by the Ligra Graph Processing Framework[4]. However, this is only almost true. To create a version of our graphs readable by iPregel, we needed to use the `SNAPToAdj` and `adjToBinary` conversion utilities provided by the Ligra Framework *and* add the edge count of the graph to the `.config` file of the resulting binary graph.

[1] A Scientist has an h-index of n if they have authored n papers that are each cited at least n times.
[2] `https://webgraph.di.unimi.it`
[3] `https://gist.github.com/iwiwi/5351417`
[4] `https://github.com/jshun/ligra`

4 Methods

4.1 Shared Memory Vertex Centric Programming & Performance

4.1.1 General Benchmarking Setup

We benchmarked the **iPregel** Vertex Centric graph processing framework [Capelli et al., 2018] against the **NetworKit** graph library [Staudt et al., 2015] as a representative comparison between vertex-centric and non-vertex-centric graph analysis. iPregel, as a framework, is designed to enable users to write their own vertex-centric algorithms. However, it comes with three already implemented algorithms, which are *Single Source Shortest Path*, *Connected Components* (only works for undirected graphs) and *PageRank*. We originally wanted to implement and benchmark further algorithms, but failed (see section 5.2) and thus decided to measure the three aforementioned ones. NetworKit, as a library, does not provide directly executable algorithms but required us to write short scripts using their given functionality to read and process graphs. The script approach enabled us to add our own time measurements (see below).
To analyze runtime and memory performance, we created a shell script based benchmarking framework that allowed us to easily execute algorithms of both iPregel and NetworKit on graphs given in a plain text edge list format with varying thread counts (see `benchmark_framework.sh` and `benchmark_template.sh` in the `utility` folder of our repository). We also added variables for customizing the list of thread counts, the number of iterations per benchmark and multithreading settings for iPregel.
For every specified algorithm-graph combination, our framework converts the edge list input to a format readable by the algorithm (only necessary for iPregel). It then first does one unmeasured run to eliminate caching inconsistencies between the first and subsequent runs. After that, the algorithm is executed the specified number of times for each thread count so that the measured values can be averaged afterwards. We used the Linux `/usr/bin/time` command to measure peak memory consumption for every run but relied on internal program measurements for time to be able to distinguish between loading, calculating and dumping time.
The iPregel framework already provided these three values using wall time[5]. Using wall time is an easy but possibly less accurate measuring technique as other processes can influence the measured runtime of the algorithms. For our setup, it seemed sufficient as we measured on a machine with only basic OS processes running apart from our measured algorithms. The iPregel output contains much more than the time information in a non-machine-friendly format, so our benchmarking framework converts this output to a CSV file with only the information we need (see `utility/iPregelBenchmarkToCSV.py`).
For our NetworKit programs, we added wall time measurements to resemble those of the iPregel framework and directly output them in a CSV format.
After running the benchmarks, we used R to average, analyze and plot the measured time and memory measurements to identify performance differences in runtime and memory consumption and the framework-specific impacts of parallelization.

4.1.2 Benchmarking Details

We executed our benchmarks on a machine featuring two Intel Xeon Gold 5220S with 18 Cores each, as well as 96GB of RAM. We thus chose to benchmark thread counts of 1, 2, 4, 8, 16, 18, 32 and 36. We ran each benchmark 100 times and averaged the results to make our measurements more comparable and accurate. We did not spot significant outliers. For iPregels multithreading settings, we chose to go for a chunk size of 256 and dynamic scheduling, which were described as the most performant configuration for a similar setup in [Capelli et al., 2019b].
In addition, we had to make some adjustments or design choices for all three of our benchmarked algorithms to make the calculations performed by iPregel and NetworKit comparable:

Connected Components As mentioned before, the iPregel implementation we benchmarked only correctly calculates Connected Components for undirected graphs. Therefore we calculated Connected Components on the undirected version of all of our graphs (containing each edge in both directions) for iPregel and NetworKit.

PageRank As this is a converging algorithm, there are multiple approaches to implement it: run the computations a predetermined number of times or stop when the overall change of the node scores between two supersteps falls below a certain threshold. The iPregel implementation uses the first (and simpler) approach, while NetworKit uses the second. To make sure both algorithms compute results of the same accuracy (and thus do the same amount of work), we first ran the NetworKit implementation and printed out the number of iterations that

[5]Wall time states how much real-time has passed between two points in a program, without regarding how much CPU time the program in question was given during that time.

were needed for convergence on that graph. We then let the iPregel implementation make the same number of iterations.

Another difference between the two implementations of PageRank is that NetworKit normalizes the node scores so that they sum up to 1. Unfortunately, implementing this in iPregel was not conveniently possible, so we wrote a separate script normalizing the iPregel results in order to be able to compare the results for equality.

SSSP While we had one given iPregel implementation for this algorithm, NetworKit offered multiple ones from which we had to choose. While we could have chosen a simple breadth-first search as all of our graphs were unweighted, we went with *Dijkstra*, the standard algorithm for SSSP, as the iPregel implementation (with minimal changes) would have been able to support weighted graphs.

Furthermore, we had to choose a start node for this algorithm. We chose to use the node with the highest score in each graph's corresponding classical node importance measure, as we expected them to reach many other nodes (in contrast to a random node that could be isolated, making the distance to every other node infinity). The resulting IDs are **398** for the *Citation* network (highest h-index), **1503** for the *Twitter* network (highest follower count), **22656** for the *stackoverflow* network (highest out-degree) and **1765703** for the *IMDb* network (most oscar nominations).

4.2 iPregel & Programmability

Additionally, to the proposed benefits of Pregel in regards to scalability and fault-tolerance, the authors of the Pregel whitepaper also claim that users find the Pregel API "intuitive, flexible, and easy to use". [Malewicz et al., 2010] Following, we will use these three features to characterize good programmability of a programming framework.
To evaluate the offered programmability by iPregel, we tried to implement algorithms for Betweenness and Closeness centrality with it. Furthermore, we planned to assess our experience for these two algorithms regarding its intuitiveness, flexibility and ease of use. We additionally evaluated the extent to which iPregel implements the Pregel API.

4.3 Alternative Node Importance Measures Analysis

We selected PageRank, Betweenness Centrality and Closeness Centrality as centrality measures to evaluate their feasibility as node importance measures in social networks.

PageRank originally was proposed as an importance measure for Web pages [Page et al., 1999]. It calculates the importance of a node based on the sum of its weighted backlinks, meaning that a link from an important node is more valuable.
Betweenness Centrality can be viewed as a measure of a node's influence on interactions between other nodes. The idea is that if a node is on the shortest path between two other nodes, it might be able to control interactions between them or gain information about their interactions. Subsequently, Betweenness Centrality is calculated for each node as the number of shortest paths that pass through it. [Wasserman and Faust, 1994]
As the name already suggests, **Closeness Centrality** is a measure of how close a node is to all other nodes in a network regarding steps needed to reach them. It is based on the idea that nodes with close connections to all other nodes can easily interact with everyone and don't need to rely on other nodes to receive information. [Wasserman and Faust, 1994] The calculation for Closeness Centrality which we used is defined for each node as the average farness (inverse distance, defined as the reciprocal of the sum of shortest paths to all other nodes).

For all datasets, we calculated PageRank, Betweenness Centrality and Closeness Centrality for each node. Based on our results, we then investigated the general feasibility of each measure as a node importance measure on each dataset. We define a node importance measure as generally *feasible*, iff it assigns interpretable scores to all nodes and distinguishes nodes from each other. To assess general feasibility we analysed the distributions of each node importance measure.
To investigate the relationship between classical and network-based node importance measures, we selected the citation network cit-HepTh and Twitter interaction network higgs-Twitter, because of their well-known and easy to calculate classical measures. Detailed descriptions for the calculation for the h-index for the cit-HepTh and the follower count for the higgs-Twitter network can be found in section 3.2. To investigate the relationship with a classical score for the respective network, we plotted and calculated the correlation between the new and classical measures.

5 Results and Discussion

5.1 Shared Memory Vertex Centric Programming & Performance

5.1.1 Performance comparison

To examine the performance differences between iPregel (vertex-centric) and NetworKit (non vertex-centric), we compared the runtime results (each averaged over 100 runs) for each of our three benchmark algorithms. We used logarithmic axes for plotting runtime against thread count so that a straight line with a downward slope from left to right is equivalent to inverse proportionality.

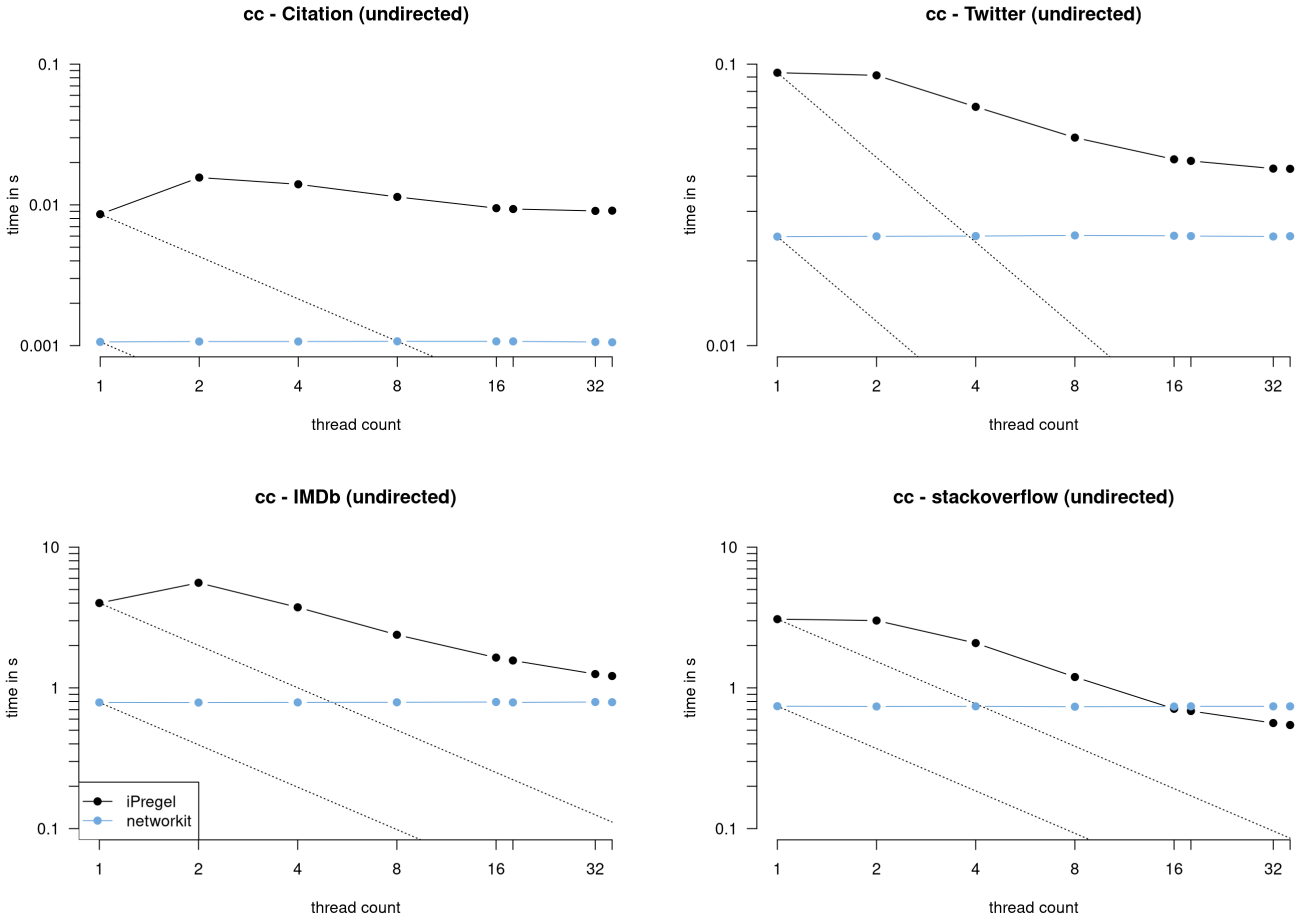


Figure 1: Runtime comparison for CC (Connected Components), dotted line is perfect parallelization.

Figure 1 shows the results for **Connected Components**. Here, NetworKit shows much better performance. We suppose that the reason for this is that NetworKit uses a simple breadth-first search, while iPregel uses a label propagation approach with far more message sending overhead. However, in contrast to NetworKit, iPregel can make use of additional threads (although there is a visible overhead for parallelization, which can be seen when going from 1 to 2 threads), matching and even superseding NetworKit's performance for higher thread counts on the stackoverflow graph. Our hypothesis is that this is due to the sparseness of the stackoverflow network, which means each vertex has fewer neighbours reducing the message sending overhead.

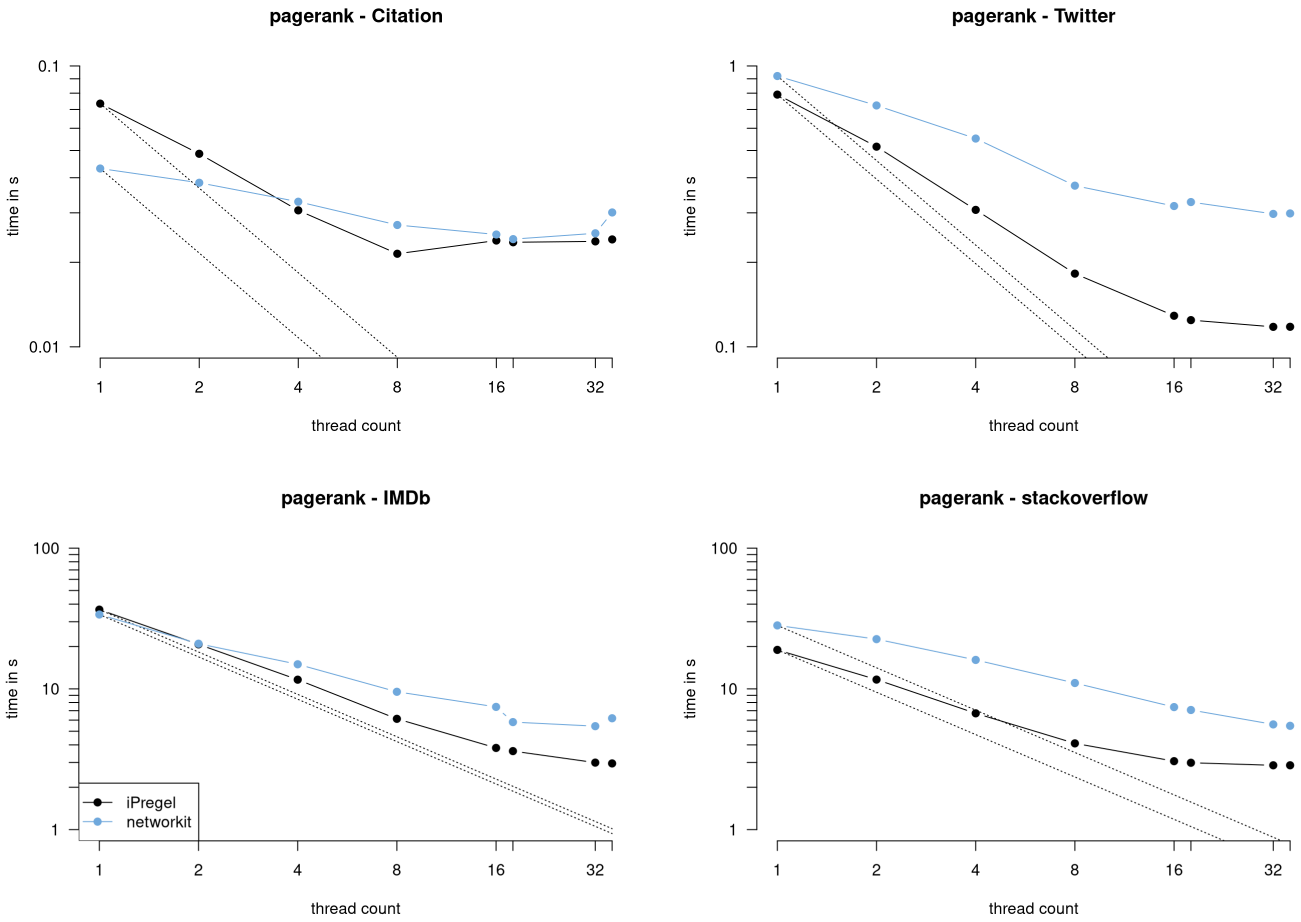
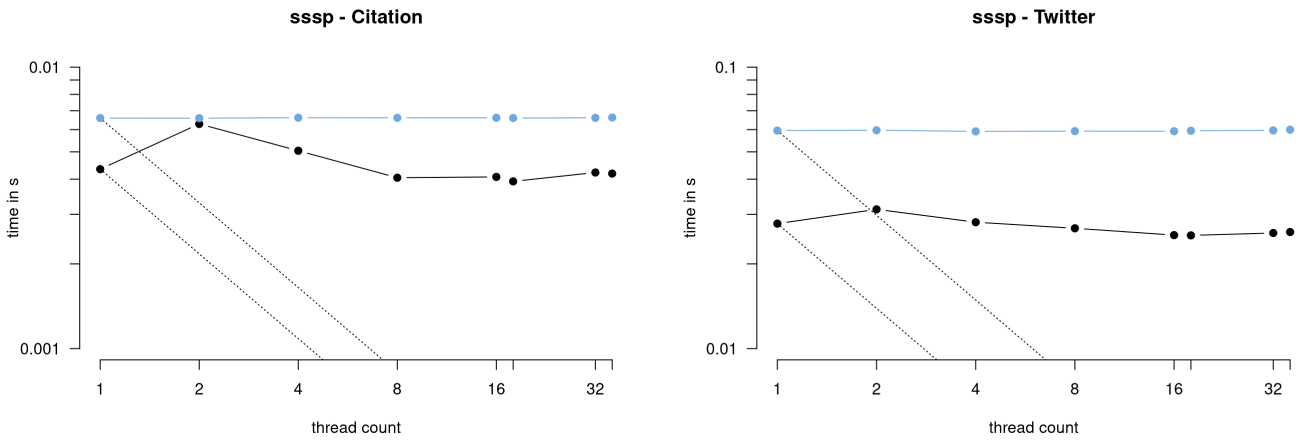


Figure 2: Runtime comparison for PageRank, dotted line is perfect parallelization.

For **PageRank**, which is an inherently vertex-centric algorithm, it does not surprise that iPregel manages to be almost consistently faster than NetworKit, as can be seen in Figure 2. However, it has to be noted that NetworKit does more work than iPregel here, as it checks for convergence each superstep and does a sum-total normalization at the end. In contrast to its Connected Components implementation, NetworKit can make use of multiple threads this time, but iPregel does so more efficiently. For both, we can observe that with higher thread counts, the performance benefit decreases as the parallelization overhead increases (this is true for all three algorithms).



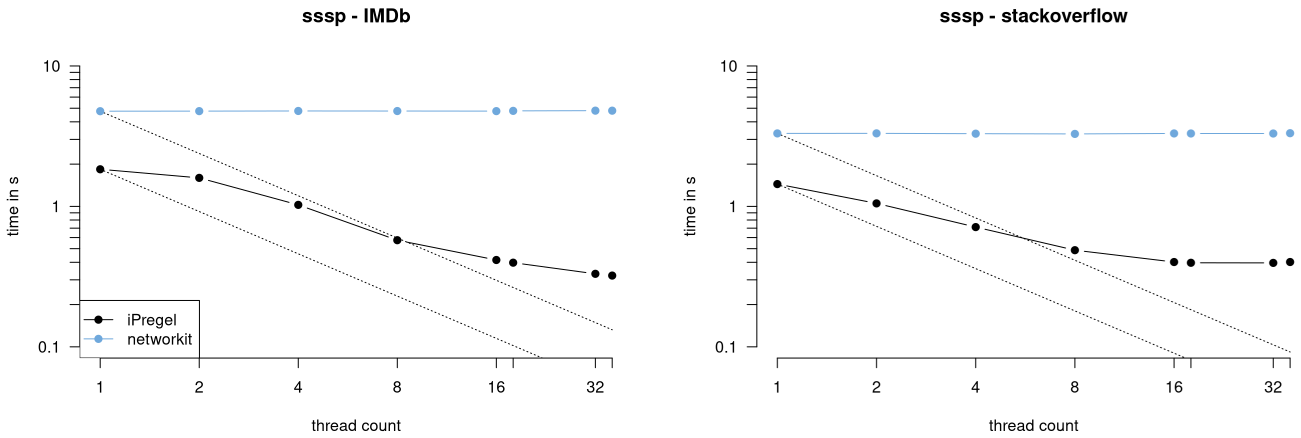


Figure 3: Runtime comparison for SSSP (Single Source Shortest Path), dotted line is perfect parallelization.

The **SSSP** results are the most interesting. Expectedly, iPregel can parallelize again while NetworKit (Dijkstra) cannot and is thus much faster on the higher thread counts. However, even on single-thread runs, iPregel is substantially faster than NetworKit even though its runtime complexity is worse in theory. The performance advantage can thus most likely not only be credited to iPregel's vertex centricity but rather to other optimizations.

5.1.2 Loading and Dumping Times

	NetworKit 1	iPregel 1	diff 1	NetworKit 36	iPregel 36	diff 36
cc - Citation (undirected)	0.0758	0.0020	-97.3%	0.0754	0.0043	-94.3%
cc - Twitter (undirected)	0.1301	0.0162	-87.6%	0.1304	0.0083	-93.6%
cc - IMDb (undirected)	60.2007	0.5571	-99.1%	60.3298	0.1321	-99.8%
cc - stackoverflow (undirected)	13.7989	0.2910	-97.9%	13.8063	0.0631	-99.5%
pagerank - Citation	0.0467	0.0222	-52.6%	0.0473	0.0265	-43.9%
pagerank - Twitter	0.1176	0.0494	-58.0%	0.1184	0.0479	-59.5%
pagerank - IMDb	60.1802	31.5836	-47.5%	60.0650	31.8820	-46.9%
pagerank - stackoverflow	10.1927	12.0200	17.9%	10.0989	12.0233	19.1%
sssp - Citation	0.0465	0.0209	-55.1%	0.0466	0.0257	-44.7%
sssp - Twitter	0.1187	0.0582	-51.0%	0.1191	0.0550	-53.8%
sssp - IMDb	60.0614	0.5479	-99.1%	60.4006	0.1349	-99.8%
sssp - stackoverflow	10.0136	8.4674	-15.4%	10.0327	8.4556	-15.7%

Table 2: Loading time (seconds) comparison for 1 and 36 threads

Table 2 shows the loading times for the different algorithm - graph combinations for minimum and maximum thread count. It is apparent that iPregels binary input format is much faster, especially on large graphs, and enables parallelized loading. We can also see that iPregel loads the undirected version of the graphs faster than the directed version, as there is no need to add the reversed in-edges (which are needed for some employed optimizations), while NetworKit is faster on the directed variants because there is less input to read.
iPregels loading disadvantage on directed graphs also explains the high loading time of the stackoverflow graph on SSSP and PageRank: It has by far the highest *directed* edge count (see Table 1).
In contrast to NetworKit, which has a general-purpose graph representation and loading functions, there is a significant difference in the loading times for the algorithms of iPregel, as it can optimize the data structures depending on what is actually needed using compilation flags.

The dumping times (shown in Table 3) provide less insights. iPregel is still faster, but not as much as before and probably mostly because it uses C instead of python for the output. Apart from that, there is no multithreading benefit observable in both networks, and the dumping is generally more or less just proportional to the amount of written data.

	NetworKit 1	iPregel 1	diff 1	NetworKit 36	iPregel 36	diff 36
cc - Citation (undirected)	0.0049	0.0024	-51.8%	0.0047	0.0028	-40.8%
cc - Twitter (undirected)	0.2100	0.0523	-75.1%	0.2132	0.0508	-76.2%
cc - IMDb (undirected)	0.9719	0.2211	-77.3%	0.9719	0.2148	-77.9%
cc - stackoverflow (undirected)	2.7982	0.6410	-77.1%	2.8023	0.6342	-77.4%
pagerank - Citation	0.0094	0.0055	-41.3%	0.0126	0.0057	-54.9%
pagerank - Twitter	0.3665	0.1689	-53.9%	0.4651	0.1611	-65.4%
pagerank - IMDb	1.7299	0.7561	-56.3%	1.7403	0.7552	-56.6%
pagerank - stackoverflow	4.5851	1.9383	-57.7%	4.6002	1.9240	-58.2%
sssp - Citation	0.0067	0.0037	-44.5%	0.0068	0.0034	-50.2%
sssp - Twitter	0.3337	0.0557	-83.3%	0.3363	0.0502	-85.1%
sssp - IMDb	1.4191	0.2267	-84.0%	1.4214	0.2200	-84.5%
sssp - stackoverflow	4.4296	0.6440	-85.5%	4.4272	0.6407	-85.5%

Table 3: Dumping time (seconds) comparison for 1 and 36 threads

5.1.3 Memory Consumption

In regards to memory consumption iPregel provides a massive advantage, as Figure 4 shows. Again, this demonstrates how optimized iPregel and its data structures are: only the information about the graph needed by the algorithm is stored. In contrast, NetworKit has a single graph class that is used for every algorithm and thus may store irrelevant information. The python environment may further increase memory demand.

5.1.4 Summary

iPregel, as a research project for vertex-centric programming with a focus on performance (and programmability), is heavily optimized and manages to offer better overall performance compared to NetworKit on two of our three tested algorithms, especially for high thread counts, but also sometimes even for single-threaded execution. In addition, its memory footprint is considerably lower. Both, however, comes at the cost of sometimes limited flexibility in regards to possible algorithms (more on that in subsection 5.2), input format and necessary preprocessing. Additionally, NetworKit did additional computation in our PageRank benchmark, leading to an unfair advantage for iPregel.

What does this mean for vertex-centric programming in general? First, our findings are not necessarily generalizable, and it would need further research to see how much of our observed performance advantage was caused by vertex centricity and how much by iPregels limiting optimizations. Also, we ran our benchmarks on comparatively small datasets, so for future work, it would be interesting to see if our observations hold with larger graphs and if parallelization still scales for even larger thread counts.
Nevertheless, our results show that vertex-centric programming introduces a natural possibility for parallel execution and thus can have performance improvement potential even on shared memory architectures compared to standard implementations.

5.2 iPregel & Programmability

iPregel offers a very simple and straightforward way to implement user-defined vertex-centric programs: A user only needs to include the iPregel header file and define

1. the data types for vertices, messages and vertex IDs
2. which information about a vertex must be accessible for the program (e.g. if the in-degree is needed)
3. a compute function which will be executed on every vertex in every superstep
4. a combine function which should merge two messages sent to the same vertex
5. a serialization function which writes relevant information about each vertex to a file

While the simplicity of this interface lowers the hurdle for creating a running vertex-centric program, it also (in our experience) massively limits the types of algorithms that can be implemented.
One of our original goals was to implement Betweenness Centrality and Closeness Centrality in a vertex-centric manner. We quickly developed an algorithm that first calculated APSP (All Pairs Shortest Paths) for the graph and

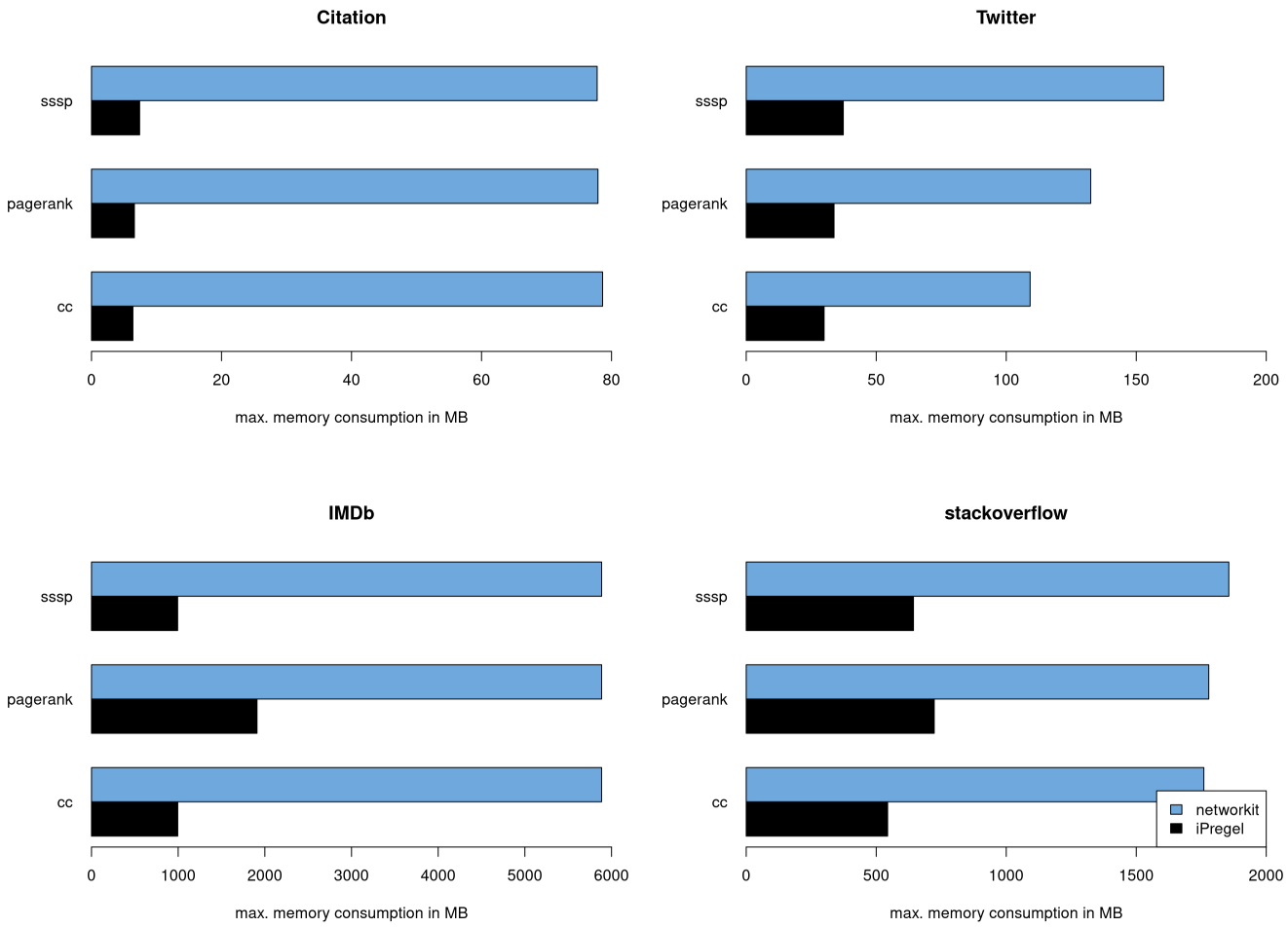


Figure 4: Memory consumption comparison for the different networks and algorithms.

then used the gathered information to generate the actual node scores. In our opinion, vertex-centric programming showed much potential, as creating an algorithm for Betweenness Centrality was effortless with the provided API. While we think our approach would have been possible to implement using the functionality described in the original Pregel paper [Malewicz et al., 2010], executing our idea failed at iPregels forced message combination and the missing implementation of aggregators. After an extended discussion with the iPregel developer, he, too, concluded that our desired algorithms (and probably many others) are currently not "idiomatically" implementable.
In total, we discovered three main shortcomings in regards to iPregels programmability, all of which are not present in the original Pregel:

1. **Forced message combination:** Our approach for calculating APSP was similar to the SSSP implementation, but instead of passing only the distance for a single, predetermined source node, we needed to send pairs of node id and distance to that node as messages. In this case, combining two messages (distances to different nodes) makes no sense but was forced by the framework. We tried to pass lists of pairs instead, making the combine operation a concatenation, but this failed due to memory management issues (iPregel is written in C).

   While iPregel used to support not combining messages, this feature was removed three years ago and recovering it would mean needing to fixing loads of compatibility issues, possibly introducing several bugs and leaving out several optimizations made in that time, so for us, this option was not feasible.

2. **No Aggregators** While it is evident that the majority of a vertex-centric program is written from the perspective of a vertex, sometimes certain values have to be aggregated across the whole graph, and decisions have to be made on them. This feature is, for example, necessary to implement NetworKit's convergence check for PageRank (accumulating the value change of all vertices and checking if the sum is below a certain threshold).

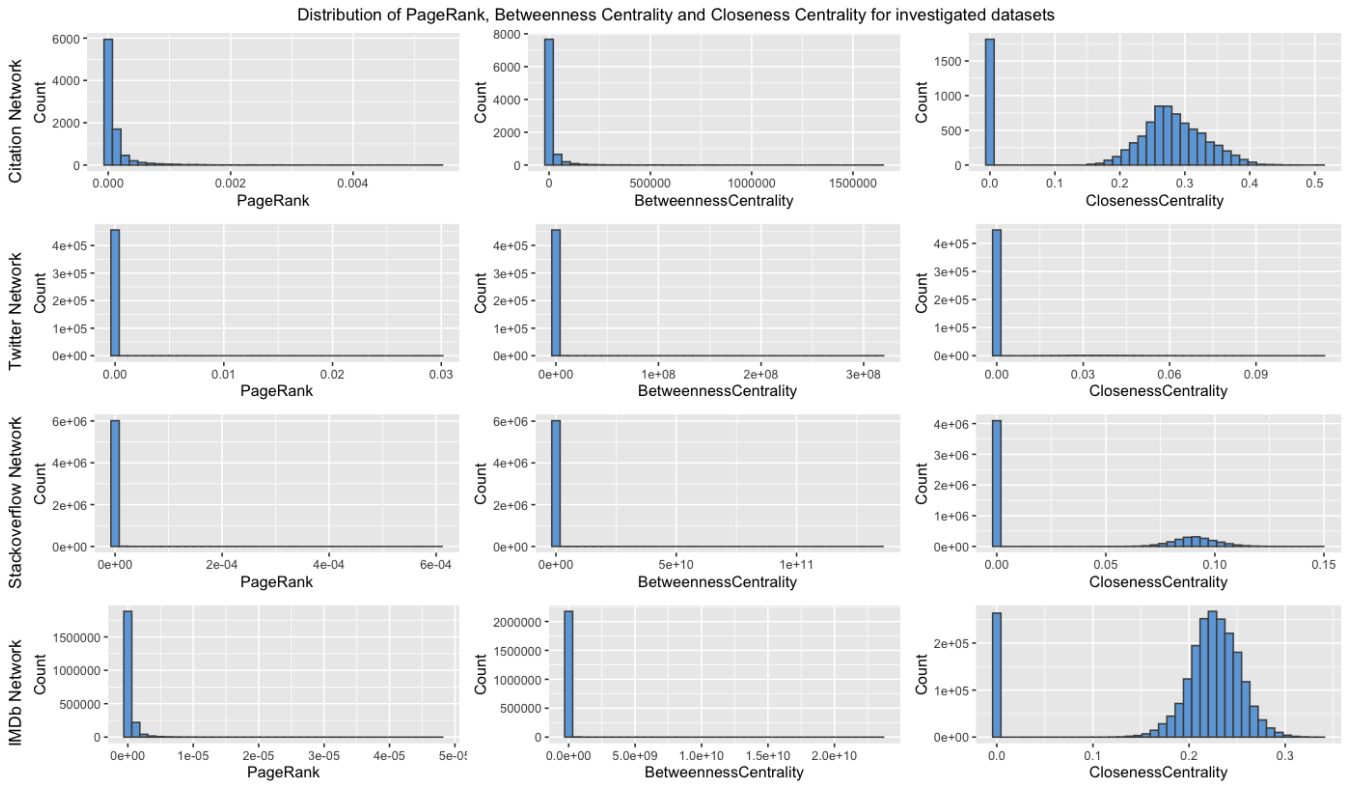


Figure 5: Distribution of PageRank, Betweenness Centrality and Closeness Centrality of investigated datasets.

Aggregators also enable multi-phase algorithms like our desired Closeness Centrality, where the vertex behaviour has to be switched at some point from calculating APSP to using the APSP results to generate the closeness score.

3. **No topology mutations:** With iPregel, it is not possible to replace groups of vertices with a single one, remove edges or do other graph mutations apart from changing edge weights. While we did not miss this feature for our node importance algorithms, it can be crucial for network analyses like community detection or other algorithms that output a graph instead of per-vertex values.

In addition, we observed some bugs during our usage of iPregel, the most notable one being that turning off optimizations led to entirely different (and wrong) results for the PageRank algorithm.
We conclude that iPregel, in its current state, is not suited for general-purpose graph analysis due to shortcomings in functionality and reliability. Regarding vertex-centric programming (VCP) in general, we had mixed experiences: Designing an algorithm for Betweenness and Closeness Centrality felt very easy and straightforward but implementing Strongly Connected Components proved to be much more difficult. Based on this, a more in-depth evaluation of VCP would be needed for a final verdict, but for now, we think that, at least for some algorithms, VCP can lead to straightforward, expressive and easy-to-write graph algorithms.

5.3 Alternative Node Importance Measures Analysis

5.3.1 Feasibility of new network-based measures

As the first step of our analysis, we investigated the general feasibility of PageRank, Betweenness Centrality and Closeness Centrality as node importance measures. We define a node importance measure as generally *feasible*, iff it assigns interpretable scores to all nodes and distinguishes nodes from each other. General feasibility explicitly does not consider semantic interpretation and validness of the assigned scores.
To evaluate general feasibility, we investigated the distributions of all three considered measures on all datasets. The distributions can be seen in Figure 5 displayed in a histogram with 40 bins each.

As a first important observation for all measures on all datasets, the biggest group of nodes gets assigned a value in the bin containing the lowest values. Furthermore, PageRank and Betweenness Centrality exhibit a massive spread

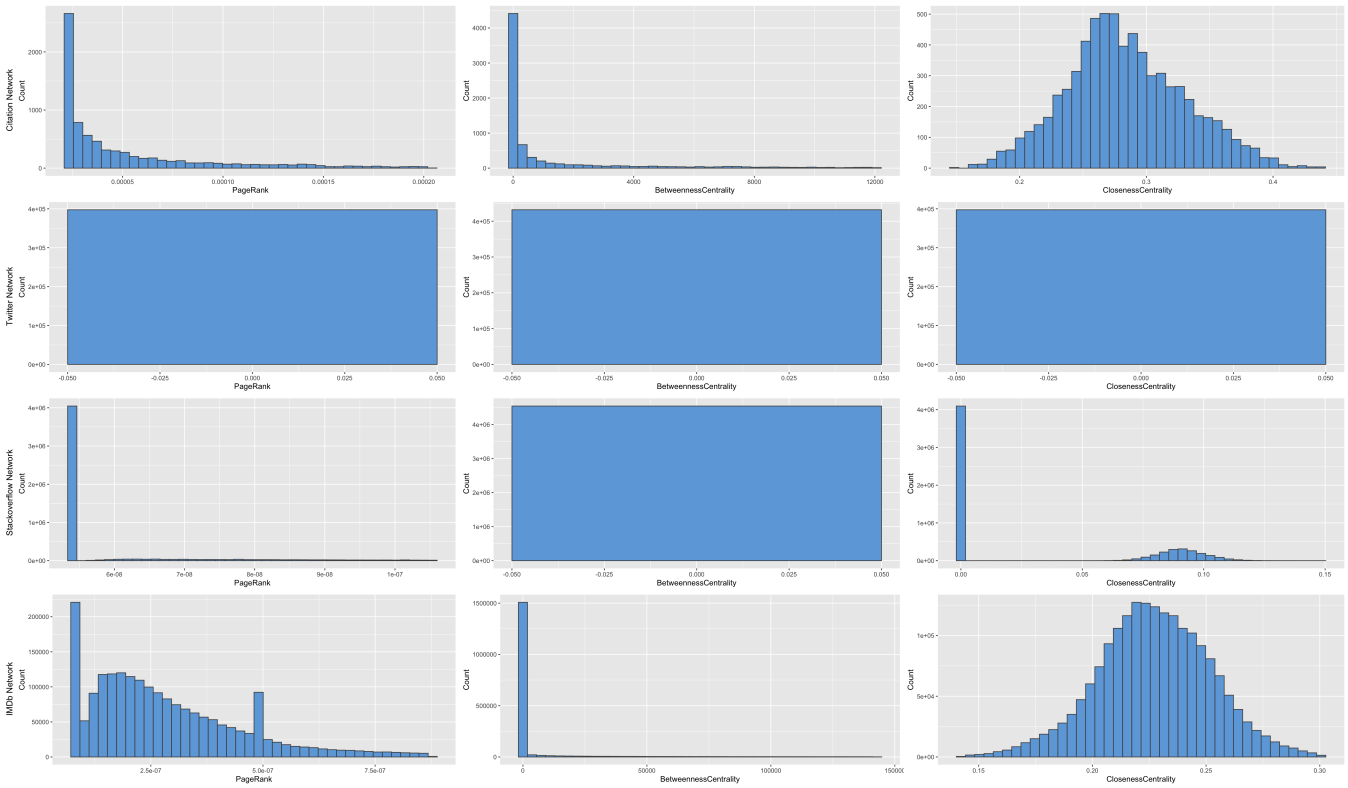


Figure 6: Interquartile range distribution of PageRank, Betweenness Centrality and Closeness Centrality of investigated datasets.

on all datasets, with what appears to depict an exponential distribution visible in the citation and IMDb networks. Closeness Centrality, in contrast, for all datasets except the Twitter network, shows a normal distribution and an outlier group consisting of the before-mentioned lowest values.

To further investigate the underlying distributions when not overshadowed by the strong lowest values group, we evaluated the Interquartile Range (IQR) distributions for all measure-network combinations as can be seen in histograms with 40 bins in Figure 6.
Quite strongly visible, there was only one value in the IQR for all three measures for the Twitter network and Betweenness Centrality on the stackoverflow network. The reason for this unexpected situation is that for all four measure-network combinations, the first three quartiles consist of the same value, namely the minimum value which the measure assigned for the network. A node importance measure that assigns the same value to 75 per cent of all nodes in a network does not meet our criteria for distinguishing nodes as 75 per cent of the nodes cannot be inter-compared. Selecting top-ranking nodes, however, was possible for all four measure-network combinations.

Interestingly, the issue of the majority of nodes being assigned the same minimum value applies for both internet social networks (Twitter, stackoverflow) in our datasets. Opposed to our other two networks (citation, IMDb actors), which depict professional interactions in a job field over a prolonged period, the Twitter network was created by displaying all interactions of Twitter users in a week based on tweets containing a specific hashtag. And while the stackoverflow network contained all interactions on stackoverflow that had happened over a period of seven years, it is far more common that people only interact once, e.g. by creating one post asking for help, than in the professional interaction networks. We assume that this lead to both internet social networks being more sparse than the professional interaction networks.

In the Twitter network 38 per cent and in the stackoverflow network 11 per cent of users interacted only once with another user and never were interacted with. That means, e.g. only asking one question in stackoverflow and never replying to somebody else's posts. Given the definition of importance of PageRank and Betweenness Centrality, which only assign importance via inleading edges to oneself / require this as a necessary precondition to be on a shortest path, all the one-interaction users naturally were assigned the lowest possible value for both measures.

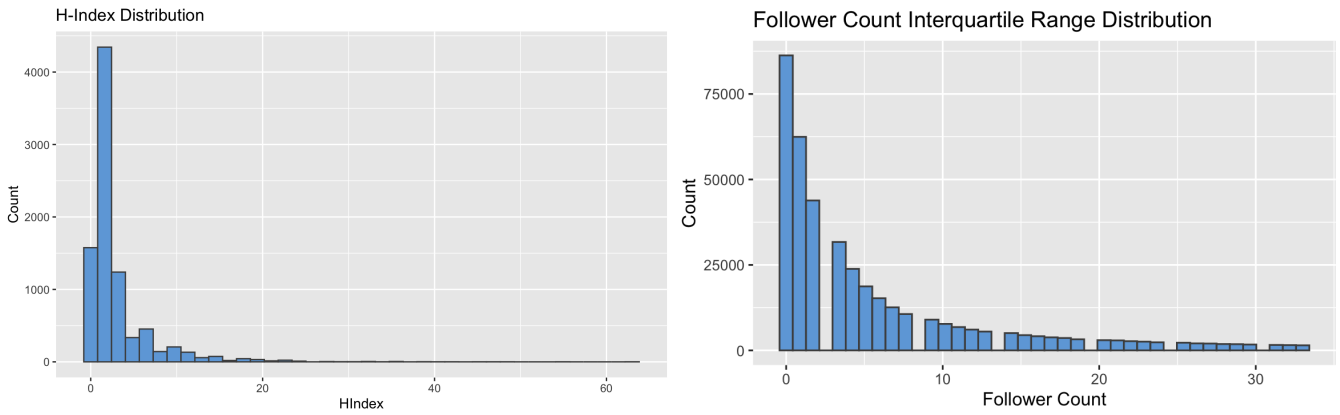


Figure 7: Distributions of classical node importance measures.

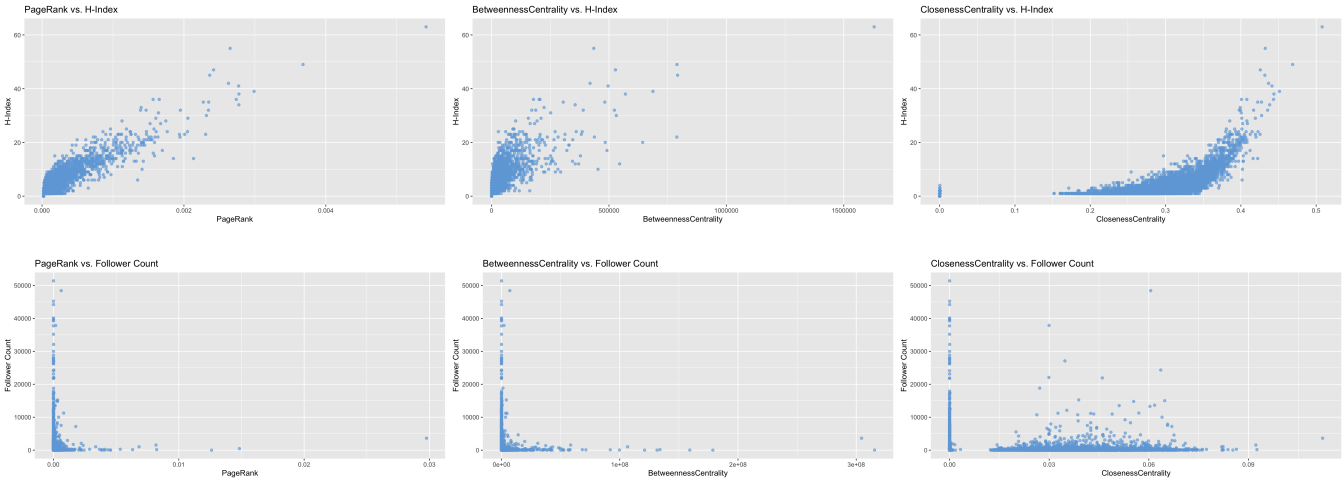


Figure 8: Correlation between classical and network-based node importance measures.

This shows the somewhat trivial notion that if one wants to rank all nodes of a network based on their interaction, the network needs to provide sufficient interaction data, meaning not being sparse, to calculate importance upon.
More interestingly, this might show a first limitation regarding domains where network-based node importance measures might be applicable. Social networks, which provide an easy way to interact only once with people one is not connected to in any real-world setting might prove to be a challenge. However, especially in huge network, it might not be essential to rank all nodes / make all nodes inter-comparable. If it is sufficient to select a certain number of top vertices, all investigated measures suffice.
Interestingly, as can be seen in Figure 7, both classical measures we investigated (h-index for the citation network, follower count for the Twitter network) show exponential distributions. Considering which ranking distribution might be appropriate for the social network one wants to rank can now become a philosophical question. An exponential distribution-based importance measure distribution works on the assumption that most people are not important / exhibit little influence, and only a numbered few actors exhibit significant influence. On the other hand, a normal distribution-based importance measure distribution works on the assumption that few people exhibit very weak or very strong influence, and most people exhibit medium influence. Which importance measures distribution might fit the ranking one wants to produce, therefore can become a design decision when working with network-based node importances measures. Future work in this area could include investigating further network-based node importance measures regarding their produced importance measure distributions and examining the distributions of the already investigated node importance measures on further datasets.

Classical measure per network	PageRank	Betweenness Centrality	Closeness Centrality
H-Index (citation network)	0.886	0.867	0.869
Follower count (Twitter network)	0.228	0.175	0.231

Table 4: Spearman correlation between classical and different network-based node importance measures.

5.3.2 Relationship of classical measures with network-based measures

As the second step of our analysis, we investigated the relationship between classical measures and network-based measures. The goal was to determine whether the new measures could provide new information about node importance in the evaluated networks. The first network we analysed was the citation network, where we compared the H-Index for each author, calculated on the underlying paper citation graph, and the new measures, calculated on the author citation graph we created by contracting paper to author nodes. As shown in Figure 8 and Table 4, which shows the Spearman correlation for all measures with the h-index, most information about the ranking created by the new measures already appears to be included in the h-index.

It appears that the heuristic of the h-index, which combines the number of papers an author has written with the number of citations each paper has received, provides a reasonable estimate for interaction importance in a scientific social network. This capability is especially interesting as the h-index can be computed linearly in the number of papers and therefore is computationally far less expensive than Betweenness Centrality and Closeness Centrality, which both possess a quadratic runtime in the number of authors (after preprocessing). It would be interesting to see whether the h-index heuristic could also be applied to estimate interaction importance in other social networks, e.g. the Twitter network, where papers could be mapped to posts of each user and citations could be mapped to interactions of other users with each post. Additionally, future work could include evaluating whether even if most information about importance is already included in the h-index, the other measures might serve to avoid known pitfalls of the h-index like being susceptible to manipulation, as is claimed by [Senanayake et al., 2015]. If that is the case, it might be valid to substitute the h-index with one of the new measures, as they appear to contain nearly the same information and might perform better in specific cases. Of course, this would require further research as our experiments were only conducted on an interaction network depicting an isolated field of study. We expect that including multiple study fields in the graph would change the existing correlation, e.g. for Betweenness Centrality, which favours nodes that connect different communities.

The second network we analysed was the Twitter network, where we compared each user's follower count, which was calculated based only on users also contained in the network and the new network-based measures calculated on the interaction graph between users based on tweet likes, mentions and replies.
As shown in Figure Figure 8 and Table 4, which shows the Spearman Correlation for all measures with the Follower Count, there appears to be no correlation. This contrasts with the high correlation between the h-index and the network-based importance measures on the citation network. The difference between the h-index and follower count, which might explain the different correlation levels, is that the h-index is calculated on interactions while the follower count is calculated on an external measure not directly linked to interaction. One might, however, expect that the follower count and, therefore, the number of people that can see and interact with ones tweets should influence the interaction level a user receives from other users. That this does not appear to be the case on the investigated network is unexpected. One factor that could have influenced the lack of correlation is that all interactions (e.g. likes, retweets) were treated the same way. Therefore, it would be interesting to investigate interaction networks that separate between the different interaction types. Additionally, the investigated network was constructed in a particular manner around a hashtag and suffers from known limitations like, e.g. the high number of nodes that only possess one out leading edge. Therefore, it would be interesting to investigate other internet social networks to see if the missing correlation between follower count and interaction levels with a user's posts can be reproduced.

In conclusion, our investigation showed that network-based importance measures appear to be generally feasible on non-sparse social networks. In comparison to classical node importance measures they can potentially offer new information, probably depending on the degree to which interaction data is already included in the classical measure.

6 Reproducing the results

1. Clone our project repository:
   `git clone https://github.com/robinwersich/DTC-2021-iPregel.git`

2. Make sure, you have the requirements described in the project `README.md` installed.

3. Inside the repository, execute `utility/setup.sh` to install the needed dependencies and download and preprocess our datasets. This can take quite a while.

4. To run the benchmarks, execute `utility/benchmark_full.sh`. It is recommended to do this on a system with enough RAM, threads and computing power. Otherwise you might want to adjust the thread counts and repetitions at the top of the file.

   The benchmark script executes SSSP, Connected Components and PageRank on all four networks for various thread counts, running each configuration 100 times and logging loading, calculating and dumping times, as well as memory consumption to the `benchmark_results` folder. The results of the algorithms are also saved in the results folder of the networks (`analyses/*/results`)

5. To obtain the additional node importance scores (Betweenness and Closeness Centrality) for the graph analysis, run `utility/benchmark_analysis.sh`. This, too, should be done on a powerful system and saves the results at the same locations as above.

6. All our data analyses were done with R in RStudio and can be reproduced via loading our analyses scripts into RStudio and executing them.

   You can find our benchmark analysis in the `benchmark_analysis` folder and our data analysis per network in a `data_analysis` folder in the respective network folder in the `analysis` folder.

7 Reflection on the project

We started this project without previous experience in graph analysis and only limited experience in setting up our own benchmarking/research project. The freedom in topic choice, goal setting and project organization was at the same time exhilarating and challenging, and in the end, we wished that we would have had more time for the actual analysis of our node importance measure results.

Based on that, our first significant learning concerns the setup and preprocessing time needed. We learned that all graph analysis libraries and processing tools we evaluated, including the libraries we chose to work with, are error-prone and should be treated with caution and double-checks. For our project, we evaluated SNAP, Web2Snap and WebGraph, and worked with NetworKit, ligra, iPregel, and a WebGraphDecoder gist. Special consideration should be given to data preprocessing, where at least the number of nodes, edges, file-endings, and node id range should be checked per dataset. Murphy's law applied in our experience, and issues with the IMDb dataset that we found out when already running our calculations blocked our analysis for some time while we had to rework some preprocessing steps and change one of our tools for preprocessing.

We had good experiences writing a collection of interconnected scripts for data preprocessing, general setup and analysis, which allowed us to run each preprocessing and analysis step in a reproducible manner. (See section 4 for a detailed description.) We did, however, not account for the time necessary for writing the benchmark framework and whole project setup scripts in our first project planning and had to adjust the project plan accordingly. Another unforeseen challenge arose when after much effort and discussion with iPregels main developer, it became apparent that it would not be possible to implement Betweenness and Closeness centrality in the current version of iPregel. This dead-end had a high impact on our project because other planned tasks like the evaluation of iPregels programmability depended on the successful implementation of Betweenness and Closeness centrality. In retrospect, we could have avoided the effort of restructuring the project in its middle by spiking its most critical parts more thoroughly. That would have included not only writing an algorithm for the intended centrality measures as a proof of concept but implementing at least one of them. The challenges in implementing our intended centrality measures connect well with the learning that graph processing libraries are often research-grade software. With iPregel, we assumed that just working with the documented interface would suffice and disregarded that potential problems with memory management or other framework internal workings could arise. As we did not know iPregel before

starting our project, more caution would have been advisable.

The last aspect where we gained experience and would approach differently in the next project is working with node clusters and performing long-running analyses. Running server-based analyses was a new experience for us, and the obtained knowledge about and comfortableness with working on node clusters is another crucial learning. However, we again underestimated the possible processing time that even middle-sized networks (approx. one million nodes) can have with algorithms, especially if run with 100 repetitions for benchmarks. Luckily our time planning worked out well in this regard, but this was not due to an accurate assessment of the needed processing time.

In conclusion, we learned a lot about setting up and following through on our own research plan. We also gained new or deepened experience with various tools and technologies, e.g. graph analysis frameworks, shell scripts for automating our workflow and RStudio for analyzing our results. Most of our key learnings focus on more accurate estimation of the time needed. We often seriously underestimated the necessary time, e.g. for preprocessing or running analyses, and learned about aspects of projects where it appears necessary to always double or triple check results and tools. We enjoyed the project and feel that we gained valuable knowledge, which we look forward to applying in future projects.

References

[Aaron Clauset and Sainz, 2016] Aaron Clauset, E. T. and Sainz, M. (2016). The colorado index of complex networks. `https://icon.colorado.edu/`.

[Brandes, 2001] Brandes, U. (2001). A faster algorithm for betweenness centrality*. *The Journal of Mathematical Sociology*, 25(2):163–177.

[Capelli et al., 2019a] Capelli, L. A., Hu, Z., Zakian, T. A., Brown, N., and Bull, J. M. (2019a). iPregel: Vertex-centric programmability vs memory efficiency and performance, why choose? *Parallel Computing*, 86:45–56.

[Capelli et al., 2019b] Capelli, L. A. R., Brown, N., and Bull, J. M. (2019b). iPregel: Strategies to Deal with an Extreme Form of Irregularity in Vertex-Centric Graph Processing. In *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 45–50, Denver, CO, USA. IEEE.

[Capelli et al., 2018] Capelli, L. A. R., Hu, Z., and Zakian, T. A. K. (2018). iPregel: A Combiner-Based In-Memory Shared Memory Vertex-Centric Framework. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, pages 1–10, Eugene OR USA. ACM.

[Gao et al., 2016] Gao, C., Wang, Z., Li, X., Zhang, Z., and Zeng, W. (2016). PR-Index: Using the h-Index and PageRank for Determining True Impact. *PLOS ONE*, 11(9):e0161755.

[Kintali, 2008] Kintali, S. (2008). Betweenness centrality : Algorithms and lower bounds.

[Leskovec and Krevl, 2014] Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

[Malewicz et al., 2010] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. page 11.

[Okamoto et al., 2008] Okamoto, K., Chen, W., and Li, X.-Y. (2008). Ranking of closeness centrality for large-scale social networks. volume 5059, pages 186–195.

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

[Pons and Latapy,] Pons, P. and Latapy, M. Computing communities in large networks using random walks. page 20.

[Senanayake et al., 2015] Senanayake, U., Piraveenan, M., and Zomaya, A. (2015). The Pagerank-Index: Going beyond Citation Counts in Quantifying Scientific Impact of Researchers. *PLOS ONE*, 10(8):e0134794.

[Staudt et al., 2015] Staudt, C. L., Sazonovs, A., and Meyerhenke, H. (2015). NetworKit: A Tool Suite for Large-scale Complex Network Analysis. *arXiv:1403.3005 [physics]*. arXiv: 1403.3005.

[Wasserman and Faust, 1994] Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press.