Scaling up Network Centrality Computations*

Alexander van der Grinten
Department of Computer Science
Humboldt-Universität zu Berlin
Berlin, Germany
avdgrinten @ hu-berlin.de

Henning Meyerhenke
Department of Computer Science
Humboldt-Universität zu Berlin
Berlin, Germany
meyerhenke @ hu-berlin.de

***Abstract*—Network science methodology is increasingly applied to a large variety of real-world phenomena. Thus, network data sets with millions or billions of edges are more and more common. To process and analyze such graphs, we need appropriate graph processing systems and fast algorithms. Many analysis algorithms have been pioneered, however, on small networks when speed was not the highest concern. Developing an analysis toolkit for large-scale networks thus often requires faster variants, both from an algorithmic and an implementation perspective.**

In this paper we focus on computational aspects of vertex centrality measures. Such measures indicate the importance of a vertex based on the position of the vertex in the network. We describe several common measures as well as algorithms for computing them. The description has two foci: (i) our recent contributions to the field and (ii) possible future work, particularly regarding lower-level implementation.

***Index Terms*—algorithmic network analysis, centrality computations, shortest paths, linear systems**

I. Introduction

Recent years have seen a proliferation of large graph-structured data sets in the order of billions of edges. Applications are numerous and come from many different scientific and industrial/commercial fields, see for example Newman [1]. Network analysis revolves around algorithmic and statistical tools to analyze such data sets – in particular with the intent to uncover non-trivial relationships between entities or groups of entities. To this end, we view a network as a graph $G = (V, E)$ with n nodes (= vertices) and $m > n$ edges in this paper.

One important class of interest for network analysis comprises *complex networks* – vaguely characterized by a nontrivial combination of randomness and structure. Prime examples are social networks, co-authorship networks, the web graph, climate networks and some biological networks such as protein interaction networks, also see [2]. Complex networks feature the small-world effect (they have a small diameter), which makes the exploitation of locality via caches challenging. The degree distribution of such networks is typically skewed – many nodes have a small neighborhood, but a few nodes have a very large neighborhood. This makes load balancing in parallel computations more difficult. These properties, along with sparsity, community structure and others, require fresh thinking about the effective usage of architectural features [3].

* This work was partially supported by grant ME 3619/3-2 within German Research Foundation (DFG) Priority Programme 1736.

One of the arguably most popular computational kernels in network analysis are based on the concept of *centrality*. Centrality measures assign to each $v \in V$ a score based on its structural importance; this allows a corresponding node ranking. (Edge centrality measures exist as well, but are omitted here.) As an example, the well-known PageRank [4] is a centrality measure for web pages. Visual illustrations of popular measures can be found at https://en.wikipedia.org/wiki/Centrality. Different applications may require different centrality measures and none is universal; thus, dozens of measures have been proposed in the literature, see [5].

Since today's data sets easily reach millions of edges, sometimes billions, the scalability[1] of analytical kernels has become a major concern. As observed by Kang *et al.* [6], "measuring centrality in billion-scale graphs poses several challenges. Many of the "traditional" definitions such as closeness and betweenness were not designed with scalability in mind." One could argue to use less compute-intensive centrality measures instead. But they are either application-specific (such as PageRank) or, as pointed out by Chen *et al.* [7], not expressive enough: "Global metrics such as betweenness centrality and closeness centrality can better identify influential nodes, but are incapable to be applied in large-scale networks (...)."

Well, not quite. The statement is true for complete exact computations, of course. But in this paper, we describe some successful attempts to scale up centrality computations in a variety of scenarios: top-k rankings, dynamic graphs, approximation, centrality improvement and group centralities. The focus will be on our recent contributions to the field, available[2] in the open-source C++/Python software NetworKit [8]. A comprehensive survey of the field is beyond the scope of this paper; nonetheless, we point to some related work as well. Moreover, we discuss interesting future work – not only from our algorithmic point of view, but also and in particular from an implementation point of view.

II. Shortest-path Centrality Measures

Information or goods (both considered in an abstract sense) often propagate through networks in shortest paths (or at least

[1] *Scalability* refers here to the capability of an algorithm to process massive data sets in reasonable time. This usually means that the time complexity is (nearly-)linear. Scalability in terms of processing elements would be an extension of this view not in the focus (but in the outlook) of this paper.

[2] https://networkit.github.io/

978-3-9819263-2-3/DATE19/©2019 EDAA

nearly-shortest paths). For applications where this usage of shortest paths reflects reality, it makes sense to assess the nodes' importance based on them. Two very popular centrality measures in this context are *betweenness* and *closeness centrality*. Both require the solution of n single-source shortest path (SSSP) searches. Generally, these searches can be accelerated by using problem-specific pruning and (in unweighted graphs) by exploiting bit parallelism [9].

A. Betweenness Centrality

The rationale behind betweenness centrality (BC) is that people in a social network (or in a company's organization chart) are important if they participate in many shortest paths. Assume that a person on many shortest paths is for some reason deleted from the network (*e. g.* due to sickness): this will impede the information flow significantly.

Formally, the BC of a node $v \in V$, $c_B(v)$, is given as

$$c_B(v) := \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{1}$$

where σ_{st} is the *number* of shortest paths between s and t and $\sigma_{st}(v)$ is the number of such paths that contain $v \neq s, t$.

1) Exact algorithms: The algorithm with the fastest asymptotic time complexity for computing all BC values is due to Brandes [10]. It performs n augmented SSSP searches and thus requires $O(n \cdot m)$ time on unweighted graphs. For sparse graphs, which are abundant in practice, this improved previous approaches by a factor of n. The main algorithmic idea is to express contributions to Eq. (1) by a recursive formula. This recursion is evaluated after each SSSP search by accumulating the contributions in a bottom-up manner in the SSSP tree. Several improvements from a practical point of view exist, both for static [11] and for dynamic graphs [12]. Yet, they do not improve the worst-case complexity of Brandes's algorithm. In fact, recent results [13] suggest that it cannot be improved under reasonable complexity-theoretic assumptions.

2) High-performance implementation techniques: As major theoretical improvements seem unlikely, one can try to speed the implementation up, *e. g.*, by utilizing parallelism and accelerators. Due to the large size of thread-local state, course-grained parallelism – in particular running multiple SSSP computations in parallel – is not well-suited for accelerators. Thus, early GPU-based algorithms for betweenness employed edge-parallel approaches that assign a thread to each edge of the graph [14] during the SSSP computation. McLaughlin and Bader [15] noticed that this is inefficient unless the current breadth-first search (BFS) level is incident to a large fraction of all edges (*e. g.* it contains a high-degree vertex in a complex network). To alleviate this, they present a hybrid algorithm that uses vertex parallelism by default. Vertex-parallel algorithms need to store the SSSP frontier explicitly, McLaughlin and Bader achieve this by implementing a queue based on atomic compare-and-swap operations. Bernaschi *et al.* [16], in turn, describe a multi-GPU algorithm based on graph partitioning and integrate various optimizations like the removal of degree-1 vertices from the graph into their GPU implementation.

For future work, note that most of those implementations only work on unweighted graphs; for weighted graphs, Dijkstra's algorithm is commonly used to compute SSSP – and that algorithm is harder to parallelize than plain BFS.

3) Approximation algorithms: Since a quadratic time complexity is too high in practice for large instances, several approximation algorithms have been proposed over the years. Different from previous sampling-based approaches (which we omit due to space constraints) is the method by Riondato and Kornaropoulos [17]: it samples node *pairs* (instead of SSSP sources) and shortest paths between them. The algorithm, let us call it RK, approximates the betweenness score of $v \in V$ as the fraction of sampled paths that contain v as intermediate node. This approach yields a probabilistic absolute approximation guarantee on the solution quality.

RK was recently improved by adaptively adjusting the number of samples and using bidirectional searches [18]. As a benchmark to compare different approximation algorithms for BC, AlGhamdi *et al.* [19] computed exact betweenness values on graphs with millions of nodes using a supercomputer.

Bergamini and Meyerhenke [20] extended RK to fully-dynamic graphs, *i. e.* graphs that can change over time by node/edge additions/deletions as well as weight changes. Such changes are fed into the graph in batches; afterwards, the centrality scores are updated. The dynamic algorithm asserts the same guarantee as the static RK algorithm: the approximated BC values differ by at most ϵ from the exact values with probability at least $1-\delta$, where $\epsilon, \delta > 0$ can be arbitrarily small constants. Running time and memory required depend on ϵ and δ. While such an absolute approximation works well for highly ranked nodes (as they have high scores), the relative position of nodes with low ranks should be treated with caution.

The main algorithmic technique to save running time compared to recomputing all results from scratch is to resample as few shortest paths as possible. Not surprisingly, the approximation achieved this way is much faster than exact approaches (also dynamic ones); it also yields significant speedups (several orders of magnitude) compared to RK. As an example, the dynamic approximation algorithm allows to track BC values for a graph with 36 million edges in a few seconds on typical workstation hardware.

We are not aware of any efficient implementations of the RK algorithm (or the algorithm from [18]) that utilize accelerators or GPUs to further improve performance.

4) Betweenness improvement problems: One way to improve your ranking in a web search engine is to have influential web pages link to yours. In more general terms, one wants to increase the centrality of a node by creating a limited amount of new edges incident to it. Restricted to BC, we considered in Bergamini *et al.* [21] (i) the problem of maximizing the betweenness score of a given node (MBI) and (ii) the problem of maximizing the ranking of a given node (MRI). The paper proves that both problems cannot be approximated very well unless $\mathcal{P} = \mathcal{NP}$. That is why it proposes a simple greedy approximation algorithm; it performs well in practice:

approximate MBI results can be computed for (most) networks with up to 10^5 edges in a matter of seconds or a few minutes.

MBI has also attracted attention from the FPT community, see in particular [22]. It would certainly be interesting to use FPT techniques in high-performance implementations as well.

B. Closeness Centrality

Let $d(x, y)$ be the distance between nodes x and y, *i. e.* the length of a shortest path between them. Then, closeness centrality (CC) of a node v, $c_C(v)$, is defined as the inverse average distance of v to all other nodes of the graph:

$$c_C(v) = \frac{n-1}{\sum_{u \in V \setminus \{v\}} d(v,u)} \quad . \tag{2}$$

Generalizations to graphs that are not (strongly) connected exist (see [23]). The textbook algorithm performs n SSSP searches and accumulates the closeness scores on the fly. This takes $O(n \cdot m)$ time in unweighted graphs – as Brandes's algorithm for BC. As with BC, there is reason to believe that an exact approach cannot be faster [23].

1) High-performance implementation techniques: In contrast to BC, however, where more complicated data structures (particularly the predecessor forest) need to be managed, computing closeness simply amounts to performing a large number of independent SSSP computations. Hence, for undirected graphs, previous results for parallel and vectorized variants of BFS carry over to closeness. Sariyüce *et al.* [24] describe a closeness algorithm that computes all SSSP results from b sources in parallel using b-bit SIMD operations. The kernel of this algorithm is a multiplication of the sparse adjacency matrix with a dense $n \times b$ matrix; this operation achieves a favorable memory access pattern.

2) Relaxing the problem: One could also try to use approximation again (as with BC) – and several algorithms have been proposed, see [25], [26]. Yet, closeness values tend to be distributed within a narrow interval [1, p. 331]. Thus, an approximation would need to have a much better accuracy (compared to BC approximation) in order to compute a trustworthy ranking. Hence, let us observe that relevant applications require only the top k nodes of the ranking (and not the exact closeness values nor the complete ranking). Such applications include visualization tasks, search engine queries and facility location in graphs. As a consequence, we proposed in Bergamini *et al.* [23] a new algorithm for selecting the k most closeness-central nodes. The main rationale is to compute (inexpensive, yet reasonably accurate) bounds on the CC values of each node. The SSSP searches from different sources are then executed in the order of the bounds. This allows to stop the process when the bounds of the remaining nodes are already worse than the top-k already found. For $k \leq 100$, the algorithm computes the top nodes in few dozens of seconds in networks with millions of nodes and edges.

Recently, the top-k algorithm was extended to dynamic networks [27]. We are not aware of GPU implementations for the top-k scenario, neither for static nor for dynamic graphs.

C. Group Closeness Centrality

Everett and Borgatti [28] extended the notion of centrality to groups of vertices: one asks how central groups of vertices are – not individually, but *as a group*. For group closeness centrality (GCC), we can define the closeness of a set S as $c_C(S) := (n - |S|)/(\sum_{v \notin S} d(S, v))$. Computing $c_C(S)$ for known groups S this way is similarly difficult as computing closeness for individual nodes. *Finding* groups with high GCC, however, leads to the $\mathcal{NP}$-hard Group Closeness Maximization (GCM) problem: find a set $S^\star \subseteq V$ of a specified size k with maximum GCC:

$$S^\star = \arg\max_{S \subseteq V} \{c_C(S) : |S| = k\}. \tag{3}$$

Chen *et al.* [29] showed that the objective function is monotonic and submodular. For such optimization problems, a simple greedy algorithm yields a $(1-1/e)$-approximation [30]. It performs k iterations and adds the vertex to the current set S that improves the marginal gain w. r. t. $c_C(S)$ the most. An implementation of this greedy algorithm with time complexity $O(n \cdot m)$ for preprocessing and $O(kn^2)$ for the loop was proposed by Chen *et al.* [29]. Its space complexity is $O(n^2)$. They also developed a faster heuristic without approximation guarantee. The limited scalability of the greedy algorithm motivated us to develop a less time- and space-consuming algorithm with the same $(1 - 1/e)$-approximation guarantee.

To this end, we adapted in Bergamini *et al.* [31] the greedy algorithm without changing its output. Memory is saved by not precomputing the pairwise distances; instead, we compute them while finding the vertex with best marginal gain with BFS techniques. To speed each search up, we use pruning and exploit submodularity. Some techniques already used for top-k closeness centrality [23] can be reused here with minor changes. One can save additional running time (at the expense of higher space complexity) if the BFS searches use bit parallelism. While graphs with 100,000 edges can be processed in fractions of a second, the largest graph in our experimental study, a social network with 117M edges, still requires 97 minutes. A further outcome is the comparison of optimal GCC values with the approximated values – for small instances that are solvable by an IP solver: on this sample all greedy results show an approximation error of less than 3%.

These results trigger several directions for future work: Can one explain the very good empirical approximation ratio with theoretical arguments? Can one exploit bit parallelism for this algorithm without increasing the space complexity? Can the algorithm be transferred to GPUs with significant performance gains (or does this hold only for certain parts such as BFS)? Finally, can one develop algorithms that scale to large graphs for other relevant group centrality measures?

III. Algebraic Centrality Measures

So far, we looked at centrality measures based in some way on *shortest* paths. This may not reflect how information is exchanged between nodes in real-world applications. It is reasonable to assume that shorter paths are preferred; but why should slightly longer paths play no role at all?

A. Electrical Closeness Centrality

Closeness centrality based on shortest-path distances has the additional drawback that its ranking is not very robust for complex networks. Since the diameter is small, all distances and thus all centrality values lie within a small interval. Moreover, adding a few edges can change the ranking significantly.

By taking paths of all lengths into account (with shorter ones being more important), *current-flow closeness centrality* (CFCC) alleviates the aforementioned drawbacks. Its ranking is more robust and the values are spread more widely [32]. This is achieved by replacing shortest-path distances by (effective) *resistance distances* [33]:

$$c_{\text{ER}}(v) := \frac{n-1}{\sum_{w \neq v} d_{\text{ER}}(v,w)}. \tag{4}$$

The (effective) resistance distance between nodes u and v, $d_{ER}(u,v)$, can be expressed as $d_{ER}(u,v) = l^{\dagger}_{uu} - 2l^{\dagger}_{uv} + l^{\dagger}_{vv}$ (up to constant factors that do not matter here), where $L^{\dagger}$ is the Moore-Penrose pseudoinverse of the Laplacian matrix L. Alternatively, one can use the solution p of the linear system $Lp = e_u - e_v$, where e_x is the canonical unit vector of dimension n, *i. e.* all entries are 0 except the one at x. Then, $d_{ER}(u,v) = p(u) - p(v)$ [32].

To compute CFCC for all nodes of G, one would need to (pseudo)invert L or solve n linear systems. In practice, inversion has cubic time complexity and leads to a dense matrix. This does not allow to scale to large networks with millions of nodes and edges, even when we want to compute the closeness of only a single node or a small subset of nodes. With this approach, in fact, computing the closeness of one node is just as expensive as computing it for all nodes.

When using the linear system approach with the fast multigrid solver LAMG [34] in its NetworKit implementation, one can estimate the closeness of a single node fairly quickly, though: as an example, for a network with 50 millions edges one needs less than 2 minutes [32]. Two inexact algorithmic approaches are proposed for this purpose: (i) a sampling-based one similar to a method for BC approximation [35] and (ii) a method based on the Johnson-Lindenstrauss transform previously used for a related edge centrality measure [36]. Our experiments indicate that the sampling-based approach is faster and preserves the ranking very well.

Recent algorithmic work has extended CFCC to the group variant [37]. For all these methods, it would be important to speed up the Laplacian solver. Algorithm theory has made tremendous progress in this regard, see [38]; so far, only few of these techniques have found their way into high-performance implementations. Early attempts [39], also with our participation [40], showed that strong theoretical worst-case results in the O-notation do not necessarily yield good practical performance. But this observation is probably ripe for reconsideration in the light of recent algorithmic progress.

B. Katz Centrality

As the second algebraic centrality measure, we consider *Katz centrality*. Let $\omega_i(v)$ be defined as the number of walks of length i that start at vertex v. Given a parameter $\alpha > 0$, the Katz centrality of v is given by:

$$c_K(v) := \mathbf{c}(v) := \sum_{i=1}^{\infty} \alpha^i \, \omega_i(v) \, . \tag{5}$$

Equivalently, in the language of linear algebra, $\mathbf{c}$ can be characterized as the solution of the linear system

$$(I - \alpha A)\mathbf{z} = \mathbf{1} \, , \qquad \mathbf{c} = \alpha A \mathbf{z}, \tag{6}$$

where A is the adjacency matrix of G, I is the identity matrix, $\mathbf{1}$ is the all-ones vector and $\mathbf{z} \in \mathbb{R}^{|V|}$ is an auxiliary variable. Indeed, previous state-of-the-art algorithms to compute $\mathbf{c}$ used to solve this system, *e. g.*, using a conjugate gradient solver. With our co-authors we presented recently the algorithm described next; it improves upon this method if Katz centrality *rankings* need to be computed (and not exact values) [41].

Definition. *Let $\ell(v)$ and $\mathbf{u}(v)$ be lower and upper bounds on $\mathbf{c}(v)$, respectively. For $\epsilon > 0$, we say that $w, v \in V$ are ϵ-separated iff the overlap of the intervals $[\ell(w), \mathbf{u}(w)]$ and $[\ell(v), \mathbf{u}(v)]$, is smaller than ϵ.*

It is easy to see that if all pairs of vertices are ϵ-separated, sorting the vertices either by ℓ or by $\mathbf{u}$ yields a correct centrality ranking, *except* that vertices w, v with $|\mathbf{c}(v) - \mathbf{c}(w)| < \epsilon$ can be ranked incorrectly. To motivate why we want to allow such errors, note that if ϵ is dropped from the definition, there are vertices that cannot be separated: Certainly, that is true for vertices w, v with identical Katz score $\mathbf{c}(w) = \mathbf{c}(v)$.

Now assume that instead of ℓ and $\mathbf{u}$, we are given sequences $\ell_i(v)$ and $\mathbf{u}_i(v)$ for each vertex v so that (i) for each i, it holds that $\ell_i(v) \leq \mathbf{c}(v) \leq \mathbf{u}_i(v)$ and (ii) $\lim_{i\to\infty} \ell_i(v) = \lim_{i\to\infty} \mathbf{u}_i(v) = \mathbf{c}(v)$. It is indeed possible to construct such sequences [41]. Our algorithm computes ℓ_i and $\mathbf{u}_i$ for each vertex of the graph and iteratively increases values of i until all vertices are ϵ-separated for a predefined ϵ. In each iteration, the algorithm sorts the vertices by their bounds to obtain a tentative ranking. Then, it checks this ranking for convergence, *i. e.*, whether all vertices are ϵ-separated. To reduce the number of such checks, the algorithm tracks a set of *active vertices* that do not have a final position in the ranking yet.

For single core CPU systems, our algorithm improves the computation time required for billions of edges down to a few minutes. Furthermore, we provide an efficient GPU implementation using the Hornet [42] graph data structure. Our results show a 10× speedup over a 20-core CPU implementation [41]. For graphs that fit into the memory of a GPU, this enables centrality computations in less than a second.

As larger graphs can only be processed on the CPU, we discuss two rather straightforward improvements to the implementation of [41]: (i) instead of representing the graph as adjacency lists, we compute ℓ_i and $\mathbf{u}_i$ on a more cache-friendly CSR representation of the adjacency matrix and

(ii) we employ a parallel multiway mergesort to obtain the ranking. The CSR representation yields a 1.3× speedup in the single-threaded case. In parallel benchmarks, the matrix representation becomes less important as a larger fraction of time is required to sort the ranking. Thus, the parallel sorting implementation results in a 2.3× speedup over the (already parallel) baseline on a 36-core machine.

IV. Conclusions and Future Work

Due to their inherent complexity, not all centrality measures can be computed exactly on large graphs with reasonable resources. Yet, as described, results can be computed much faster when changing the task slightly and/or using high-performance implementations. This way, many popular centrality computations can be executed reasonably quickly, even on large-scale graphs. Important challenges remain and shall be mentioned next in some detail.

Massive graphs: If one wants to model the human connectome (a map of neural connections in the brain), graphs with ca. 10^{10} nodes and 10^{14} edges have to be processed [43]. This number of edges is well beyond current capabilities in terms of space and time – also when considering distributed computation because then time becomes an issue. How can we scale to processing such large graphs efficiently? Clearly, the algorithms used for this purpose need to run in (nearly-)linear time. Moreover, such graphs are unlikely to fit into one machine's memory for the near future. Thus, partitioning needs to distribute subgraphs onto compute nodes in a careful manner. This partitioning step is still challenging for complex networks, but recent progress is promising [44], [45].

High-performance techniques: Of course, being able to store large graphs and to perform some calculations alone is not sufficient. The implementations need to use high-performance techniques if the computations shall terminate in reasonable time. Distributed graph frameworks come with a certain overhead that makes (or at least made them in 2015 [46]) significantly less efficient than shared-memory approaches. A stronger convergence between high-performance and high-productivity frameworks is thus desirable. Emerging hardware architectures tailored to data analytics tasks such as Emu[3] may become a game changer in this regard.

While efforts exist to utilize techniques such as accelerators, vectorization and SIMD for network analysis (*e. g.* those mentioned in this paper), such optimizations are mostly implemented as specialized programs or as small frameworks, but rarely in feature-rich general-purpose network analysis toolkits. An emerging technology to make high-performance graph operations more widely available is the GraphBLAS standard [47], [48]. This standard defines an API to perform graph operations in the language of linear algebra. Kernels such as SSSP can be expressed in this language and thus benefit from highly-optimized GraphBLAS implementations.

[3]http://www.emutechnology.com/

Usability: While domain experts do care about performance, they often care even more about usability. Hence, the integration of high-performance techniques into general frameworks should not decrease a tool's usability (significantly). NetworKit's approach of combining C++ for fast algorithms and Python for usability (also done by other toolkits) is an attempt in this direction. Python is relatively easy to learn and offers a rich ecosystem of interoperable modules. But that does not mean it is a silver bullet for all scientific communities.

Data types: From an algorithmic point of view, the abstract data type *graph* is not sufficient for all applications. As an example, hypergraphs are more accurate in capturing non-binary relationships. Graph attributes, if taken into account by an algorithm, usually increase the computational complexity as well. In graphs arising from experiments with measurement errors, edges may exist only with a certain probability. These changes may have dramatic effects on the definition and computation of centrality measures, in particular in the context of high-performance implementations.

Personal conclusion: Let us stress that we are certain (but maybe not always aware) that very promising progress has been reached in recent years on any individual aspect mentioned above. Parts of this progress have been mentioned in this paper (we admit a deliberate bias). If you already solved future work items we pointed out, please let us know.

In our opinion a major gap to be filled by the research community is the elegant combination of the improvements already reached. Actually, this would mean to integrate several diverse research communities, which is ambitious. But would it not be great if these individual improvements could be brought together in one (or a few interoperable) powerful, yet convenient tool(s) with a rich set of efficient algorithms and high-performance software/hardware support?

Acknowledgment

We thank the co-authors of our works described in this paper for their indispensable contributions. Moreover, we thank all contributors to NetworKit, see https://networkit.github.io/credits.html.

References

[1] M. Newman, *Networks*. OUP Oxford, 2nd ed., 2018.
[2] K. Erciyes, *Complex Networks. An Algorithmic Perspective*. CRC Press, 2015.
[3] Y. Zhang, V. Kiriansky, C. Mendis, S. P. Amarasinghe, and M. Zaharia, "Making caches work for graph analytics," in *2017 IEEE International Conference on Big Data, BigData 2017*, pp. 293–302, 2017.
[4] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 107 – 117, 1998. Proceedings of the Seventh International World Wide Web Conference.
[5] P. Boldi and S. Vigna, "Axioms for centrality," *Internet Mathematics*, vol. 10, no. 3-4, pp. 222–262, 2014.
[6] U. Kang, S. Papadimitriou, J. Sun, and H. Tong, "Centralities in large networks: Algorithms and observations," in *Proceedings of the 2011 SIAM International Conference on Data Mining*, pp. 119–130, SIAM, 2011.
[7] D. Chen, L. Lü, M.-S. Shang, Y.-C. Zhang, and T. Zhou, "Identifying influential nodes in complex networks," *Physica a: Statistical mechanics and its applications*, vol. 391, no. 4, pp. 1777–1787, 2012.
[8] C. L. Staudt, A. Sazonovs, and H. Meyerhenke, "NetworKit: A tool suite for large-scale complex network analysis," *Network Science*, vol. 4, no. 4, pp. 508–530, 2016.

[9] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*, pp. 349–360, ACM, 2013.

[10] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[11] R. Puzis, Y. Elovici, P. Zilberman, S. Dolev, and U. Brandes, "Topology manipulations for speeding betweenness centrality computation," *Journal of Complex Networks*, vol. 3, no. 1, pp. 84–112, 2015.

[12] E. Bergamini, H. Meyerhenke, M. Ortmann, and A. Slobbe, "Faster betweenness centrality updates in evolving networks," in *SEA 2017, London, UK*, vol. 75 of *LIPIcs*, pp. 23:1–23:16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[13] M. Borassi, P. Crescenzi, and M. Habib, "Into the square: On the complexity of some quadratic-time solvable problems," *Electronic Notes in Theoretical Computer Science*, vol. 322, pp. 51 – 67, 2016. Proceedings of ICTCS 2015, the 16th Italian Conference on Theoretical Computer Science.

[14] Y. Jia, V. Lu, J. Hoberock, M. Garland, and J. C. Hart, "Edge v. node parallelism for graph centrality metrics," in *GPU Computing Gems Jade Edition*, pp. 15–28, Elsevier, 2011.

[15] A. McLaughlin and D. A. Bader, "Scalable and high performance betweenness centrality on the GPU," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'14*, pp. 572–583, 2014.

[16] M. Bernaschi, G. Carbone, and F. Vella, "Scalable betweenness centrality on multi-gpu systems," in *Proceedings of the ACM International Conference on Computing Frontiers, CF'16*, pp. 29–36, 2016.

[17] M. Riondato and E. M. Kornaropoulos, "Fast approximation of betweenness centrality through sampling," *Data Mining and Knowledge Discovery*, vol. 30, no. 2, pp. 438–475, 2016.

[18] M. Borassi and E. Natale, "KADABRA is an adaptive algorithm for betweenness via random approximation," in *24th Annual European Symposium on Algorithms, ESA'16*, vol. 57 of *LIPIcs*, pp. 20:1–20:18, 2016.

[19] Z. AlGhamdi, F. Jamour, S. Skiadopoulos, and P. Kalnis, "A benchmark for betweenness centrality approximation algorithms on large graphs," in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, SSDBM'17*, pp. 6:1–6:12, 2017.

[20] E. Bergamini and H. Meyerhenke, "Approximating betweenness centrality in fully dynamic networks," *Internet Mathematics*, vol. 12, no. 5, pp. 281–314, 2016.

[21] E. Bergamini, P. Crescenzi, G. D'angelo, H. Meyerhenke, L. Severini, and Y. Velaj, "Improving the betweenness centrality of a node by adding links," *Journal of Experimental Algorithmics*, vol. 23, pp. 1.5:1–1.5:32, Aug. 2018.

[22] C. Hoffmann, H. Molter, and M. Sorge, "The parameterized complexity of centrality improvement in networks," in *SOFSEM 2018: Theory and Practice of Computer Science - 44th International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 111–124, 2018.

[23] E. Bergamini, M. Borassi, P. Crescenzi, A. Marino, and H. Meyerhenke, "Computing top-k closeness centrality faster in unweighted graphs," in *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016*, pp. 68–80, SIAM, 2016.

[24] A. E. Sariyüce, E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Regularizing graph centrality computations," *Journal of Parallel and Distributed Computing*, vol. 76, pp. 106–119, 2015.

[25] D. Eppstein and J. Wang, "Fast Approximation of Centrality.," *Journal of Graph Algorithms and Applications*, pp. 39–45, 2004.

[26] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck, "Computing classic closeness centrality, at scale," in *Proceedings of the second ACM conference on Online social networks, COSN 2014*, pp. 37–50, ACM, 2014.

[27] P. Bisenius, E. Bergamini, E. Angriman, and H. Meyerhenke, "Computing top-k closeness centrality in fully-dynamic graphs," in *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018*, pp. 21–35, SIAM, 2018.

[28] M. G. Everett and S. P. Borgatti, "The centrality of groups and classes," *Journal of mathematical sociology*, vol. 23, no. 3, pp. 181–201, 1999.

[29] C. Chen, W. Wang, and X. Wang, "Efficient maximum closeness centrality group identification," in *Databases Theory and Applications - 27th Australasian Database Conference, ADC 2016*, vol. 9877 of *Lecture Notes in Computer Science*, pp. 43–55, Springer, 2016.

[30] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[31] E. Bergamini, T. Gonser, and H. Meyerhenke, "Scaling up group closeness maximization," in *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018*, pp. 209–222, SIAM, 2018.

[32] E. Bergamini, M. Wegner, D. Lukarski, and H. Meyerhenke, "Estimating current-flow closeness centrality with a multigrid laplacian solver," in *2016 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing, CSC 2016*, pp. 1–12, SIAM, 2016.

[33] U. Brandes and D. Fleischer, "Centrality measures based on current flow," in *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2005*, vol. 3404 of *LNCS*, pp. 533–544, Springer, 2005.

[34] O. E. Livne and A. Brandt, "Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. B499–B522, 2012.

[35] U. Brandes and C. Pich, "Centrality estimation in large networks," *International Journal of Bifurcation and Chaos*, vol. 17, no. 7, pp. 2303–2318, 2007.

[36] C. Mavroforakis, R. Garcia-Lebron, I. Koutis, and E. Terzi, "Spanning edge centrality: Large-scale computation and applications," in *Proceedings of the 24th International Conference on World Wide Web, WWW 2015*, pp. 732–742, ACM, 2015.

[37] H. Li, R. Peng, L. Shan, Y. Yi, and Z. Zhang, "Current flow group closeness centrality for complex networks," *CoRR*, vol. abs/1802.02556, 2018.

[38] R. Kyng and S. Sachdeva, "Approximate gaussian elimination for laplacians - fast, sparse, and simple," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pp. 573–582, IEEE Computer Society, 2016.

[39] K. Deweese, J. R. Gilbert, G. L. Miller, R. Peng, H. R. Xu, and S. C. Xu, "An empirical study of cycle toggling based laplacian solvers," in *2016 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing, CSC 2016*, pp. 33–41, SIAM, 2016.

[40] D. Hoske, D. Lukarski, H. Meyerhenke, and M. Wegner, "Engineering a combinatorial laplacian solver: Lessons learned," *Algorithms*, vol. 9, no. 4, p. 72, 2016.

[41] A. van der Grinten, E. Bergamini, O. Green, D. A. Bader, and H. Meyerhenke, "Scalable katz ranking computation in large static and dynamic graphs," in *26th Annual European Symposium on Algorithms, ESA 2018*, vol. 112 of *LIPIcs*, pp. 42:1–42:14, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[42] F. Busato, O. Green, N. Bombieri, and D. Bader, "Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs," in *IEEE Proc. High Performance Extreme Computing (HPEC)*, (Waltham, MA), 2018.

[43] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541.

[44] G. M. Slota, S. Rajamanickam, K. D. Devine, and K. Madduri, "Partitioning trillion-edge graphs in minutes," in *2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017*, pp. 646–655, IEEE Computer Society, 2017.

[45] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2625–2638, 2017.

[46] J. Koch, C. L. Staudt, M. Vogel, and H. Meyerhenke, "An empirical comparison of big graph frameworks in the context of network analysis," *Social Network Analysis and Mining*, vol. 6, no. 1, pp. 84:1–84:20, 2016.

[47] J. Kepner, P. Aaltonen, D. A. Bader, A. Buluç, F. Franchetti, J. R. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. G. Mattson, and J. E. Moreira, "Mathematical foundations of the graphblas," in *HPEC 2016*, pp. 1–9, IEEE, 2016.

[48] A. Buluç, T. Mattson, S. McMillan, J. E. Moreira, and C. Yang, "Design of the graphblas API for C," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017*, pp. 643–652, 2017.