Approximation of interactive betweenness centrality in large complex networks

Sebastian Wandelt, Xing Shi, and Xiaoqian Sun*

School of Electronic and Information Engineering, Beihang University, 100191 Beijing, China and National Engineering Laboratory for Integrated Transportation Big Data Center, 100191 Beijing, China

The analysis of real-world systems through the lens of complex networks often requires a node importance function. While many such views on importance exist, a frequently-used global node importance measure is betweenness centrality, quantifying the number of times a node occurs on all shortest paths in a network. This centrality of nodes often significantly depends on the presence of nodes in the network; once a node is missing, e.g. due to a failure, other nodes’ centrality values can change dramatically. This observation is, for instance, important when dismantling a network: Instead of removing the nodes in decreasing order of their static betweenness, recomputing the betweenness after a removal creates tremendously stronger attacks, as has been shown in recent research. This process is referred to as interactive betweenness centrality. Nevertheless, very few studies compute the interactive betweenness centrality, given its high computational costs, a worst-case runtime complexity of $O(N^4)$ in the number of nodes in the network.

In this study, we address the research questions, whether approximations of interactive betweenness centrality can be obtained with reduction of computational costs; and how much quality/accuracy needs to be traded in order to obtain a significant reduction. At the heart of our interactive betweenness approximation framework, we use a set of established betweenness approximation techniques, which come with a wide range of parameter settings. Given that we are interested in the top-ranked node(s) for interactive dismantling, we tune these methods accordingly. Moreover, we explore the idea of batch removal, where groups of top-k ranked nodes are removed before recomputation of betweenness centrality values. Our experiments on real-world and random networks show that specific variants of the approximate interactive betweenness framework allow for a speedup of two orders of magnitude, compared to the exact computation, while obtaining near-optimal results. This work contributes to the analysis of complex network phenomena, with a particular focus on obtaining scalable techniques.

I. INTRODUCTION

Complex network theory provides powerful tools to understand the structures and dynamics of many complex systems. Essentially, these systems are being modelled as nodes representing entities and links representing dependencies between entities. Much research effort has been spent on understanding different types of critical infrastructure systems, e.g. energy [1, 2], communication [3, 4], air transportation [5–8], railway [9], as well as social networks [10]. The phenomena and processes analyzed on these networks varies by study, including resilience analysis [11, 12], delay/information spreading [13–15], growth pattern analysis and many others. Nevertheless, at the heart of many analysis tasks is the problem of identifying node importance, i.e. a quantification of the relative value of a node in a network. Indeed, it is significant to identify the extremely important nodes which maintain the structure and function of the network.

These node importance values vary for two reasons. First, the importance can be measured regarding different perspectives of importance; preferring local vs global or topological vs flow-like views. Depending on the chosen view, many different node centrality measures have been proposed, including degree centrality, closeness centrality [16], eigenvector centrality [17], Katz centrality [18] and betweenness centrality [19]. Second, the importance of a node often depends significantly on the presence of other nodes in the network. For a pair of nodes with redundant function, e.g., regarding propagation, one node can become significantly more important in the absence of the other node. This effect is visualized in Figure 1. Initially, node 9 is not important in the network. However, once node 14 fails, the majority of flow in the network is routed via node 9, since all flows have to go through the remaining path on the right-hand side. Accordingly, a very small change in the network, here referring to the failure of a node, can change the node importance significantly.

Existing methods usually do not take into account this dependency of node importance values, mainly because of limited computational resources. For instance, computing exact betweenness centrality values of each node in a

* Corresponding author: qian.sun@gmail.com
FIG. 1. Static betweenness attack and interactive betweenness attack on an example network. The node with the highest betweenness is highlighted with larger node size. The red node represents the attacked node in each subplot.

network, has a worst-case time complexity cubic in the number of nodes, since essentially, all pairs of shortest path between all nodes have to be computed [20]. Computing the interactive betweenness centrality requires to recompute the betweenness centrality after each node removal, increasing the worst-case time complexity to being quartic in the number of nodes in the network, i.e. $O(N^4)$. Such a high computational complexity inhibits computations on even medium-sized networks, given that increasing the size of a network by a factor of 10, will increase the required computational resources by a factor of 10,000. While static betweenness centrality computations can be speed-up significantly by parallelization [21], interactive betweenness centrality cannot be further accelerated, given the dependency of choices between each attack step: the subnetwork at step $i + 1$ is only determined, once the to-be-removed node at step $i$ is fixed.

In this study, we aim to explore possibilities for computing an approximation of the interactive betweenness centrality for larger networks. To achieve this goal, we devise an estimation framework. We exploit betweenness approximation techniques for selecting outstandingly important nodes in a network. There are several widely-used static betweenness approximation methods, which come with whole range of parameters. Moreover, in order to avoid re-computing the approximate betweenness at each iteration, we select a number of outstanding nodes (not only one) on-the-fly. Experiments on random and real-world networks show that this strategy computes rankings very similar to those obtained by exact interactive betweenness computation. Moreover, experiments on network dismantling show that the results obtained by approximation of interactive betweenness are close to those of interactive betweenness, but at much lower runtime requirements. Our work contributes to the analysis of complex network phenomena, with a particular focus on obtaining scalable techniques.

II. METHODS

A. The overall framework

We devised an interactive betweenness approximation framework consisting of a set of static betweenness approximation algorithms and different selections of k-batch removal (remove k nodes with high betweenness before recomputation). The exact interactive computation is recomputing betweenness values after removal of the Top-1
node. However, the time complexity of static betweenness computation is $O(N \cdot E)$, where $N$ is the number of nodes and $E$ is the number of edges in the network, which is prohibitive for large networks and makes the interactive computation more expensive. To reduce the computational costs, we exploit approximation methods since such methods can make trade-offs between speed and identification of high betweenness nodes. Note that, the identification of the highest betweenness node is the core part of interactive computation. Moreover, we also considered selection of $k$ (i.e., choice of how many nodes to remove in each iteration): Instead of removing a single node with the highest betweenness, batch removal reduces the number of iterations of interactive computation. Based on the above ideas, our framework has two core parts:

1. **Static betweenness estimation**: Compute estimated betweenness values of all nodes in the current GCC (giant connected component) of the network.

2. **Selection of batch removal**: Obtain Top-$k$ ranked nodes and remove them from the network, then go back to 1).

In part 1), the approximation algorithms estimate betweenness values of each node in the current GCC. The accuracy of approximation affects the quality of such interactive computation: If the approximation method cannot identify the Top-$k$ nodes correctly, it will lead to continuous errors in subsequent iterations, which propagate and often become worse with an increasing number of nodes. Therefore, we need to select approximation methods with nice trade-offs between quality and runtime. In part 2), the selection of the parameter $k$ is also worth considering. On the one hand, if we choose small $k$, we will get better quality. One of the most extreme choices is setting $k = 1$, that is, recomputing betweenness after each node’s removal, which will be extremely time consuming but can get exact results. On the other hand, if we set $k$ very large, we can reduce runtime for the price of deteriorated quality. In the best case, the $k$ value is chosen adaptively in each iteration, as there may be only one or many high betweenness nodes in the current GCC. To sum up, our interactive betweenness approximation framework focuses on the selections of approximation methods together with the number and size of batch removal.

### B. Static Betweenness Approximation

The existing algorithms for betweenness approximation compute an estimation of the static betweenness of all nodes in the network. Since all the approximation methods are based on Brandes’ algorithm, we revisit this algorithm first. For a node pair $(s, t)$, [20] defines the pair-dependency on node $v$, denoted by $\delta_{st}(v)$ and dependency of node $s$ on node $v$, denoted by $\delta_{s*}(v)$ as:

$$
\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} \quad \text{and} \quad \delta_{s*}(v) = \sum_{t \in G} \delta_{st}(v)
$$

(1)

In addition, Brandes proved that $\delta_{s*}(v)$ obeys:

$$
\delta_{s*}(v) = \sum_{t : v \in P_{st}(t)} \frac{\sigma_{sv}}{\sigma_{st}} (1 + \delta_{s*}(t))
$$

(2)

where $P_{st}(t)$ represents all parents of $t$ on the breadth-first-search (BFS) from $s$. Based on these, the betweenness value of $v$ $B(v)$ can be computed by $B(v) = \sum_{s \in G, s \neq v} \delta_{s*}(v)$. That is, given a network with $N$ nodes and $E$ edges, a single BFS from one source node $s$ can compute the dependency of each node which takes $O(E)$ time. To obtain betweenness of all nodes, each node of the network should be set as a source node and it requires $N$ iterations of BFS. In total, the computation of exact static betweenness of all nodes needs $O(N \cdot E)$ time, which is quite expensive for large networks. Besides, for dense networks with $E \approx N^2$, as the worst case, the time complexity is $O(N^3)$.

To reduce the computational cost, approximation methods compute a subset of node dependencies or pair dependencies instead of the set of all dependencies required by the exact computation. Different strategies for selecting the subset constitute several approximating methods. In general, there are three classifications:

1. **Pivots sampling**: Such methods conduct BFS from a subset of source nodes, called pivots, and compute node dependencies on each node from selected pivots.

2. **Node pairs sampling**: Instead of considering node dependencies, such methods sample pairs of nodes and compute the pair dependencies on each node from selected node pairs.

3. **Bounded BFS**: Such methods change the stop condition of BFS and only consider a subset of shortest paths.
Besides these three classifications, there are also some recent methods for betweenness approximation, including sparse-modeling based method [22], MPI-based adaptive sampling method [23] and GNN-based method [24]. More details for different static approximating methods and parameter settings are in Appendix A.

C. Choosing the size of batch removal

In this subsection, we describe more details on how to determine k based on current GCC in each iteration. If k is small, few nodes are removed from current GCC, which requires more iterations and higher computational costs. On the contrary, if k is large, the computational costs will be reduced but the quality decreases since many removed nodes have lost their importance. Therefore, we need to make a trade-off between quality and speed. Note that, it is more reasonable to choose an adaptive k value based on the number of particularly central nodes in each iteration. Firstly, we need to roughly estimate the range of k for different networks. We selected k = 1 and conducted experiments of the interactive exact betweenness computation on 48 real-world networks with diverse sizes. We visualize the distribution of $\rho_{50\%}$ (the number of nodes that need to be removed to get 50% GCC reduction) in Figure 2. As the left subplot shows, some larger networks can be cut into 50% with removal of few nodes (e.g., removing no more than 10 nodes can get 50% reduction on a network with size $> 10,000$). Besides, the distribution of $\rho_{50\%}$ in right subplot indicates that removal of no more than 50 nodes can cause 50% GCC reduction on many networks. When being interested in fixed-size batch removal, we set $k \in [1, 2, 4, 8, 16]$.

![FIG. 2. Distribution of $\rho_{50\%}$ for 48 real-world networks.](image)

Figure 3 shows the distribution of betweenness values under different attack strategies. We can see that in I4 (Interactive 4th attack) and IR (Interactive remainder), there are 2 nodes with high betweenness (e.g., node 2 and 3 with betweenness value = 0.5 in IR) and these two nodes are of the same importance. we can remove both of them in one iteration to break up the GCC. In I3 (Interactive 3rd attack), there is only one node (i.e. node 5) with high betweenness value (0.5), that is, there is only one particularly central node. For such an outstandingly important node, it is reasonable to set $k = 1$ and only remove the single node from GCC. Inspired by the example network, we, in addition, consider setting k to be the number of nodes with betweenness $\geq 0.5$ and make it adaptive in the range $[1, 2, 4, 8, 16]$. Besides, we also consider the case of setting k to be the number of nodes with betweenness $\geq$ (average + standard deviation of betweenness values). Besides, we can remove certain percentage of nodes in each iteration. For the remaining experiments, we selected 1%, 5%, 10% and 20%. To sum up, we determine k value in each iteration based on the distribution of betweenness values of nodes in current GCC. Table I shows an overview of k settings for batch removal.

D. Measures for comparison

**Accuracy**: Given an approximation algorithm and certain k setting, the output of our framework is a ranking of nodes from higher interactive betweenness to lower interactive betweenness. To analyze the approximated ranking, we considered four aspects:
FIG. 3. Distribution of betweenness values of sample network in Figure 1.

TABLE I. Overview of k settings. \(aB\): betweenness value of one node. \(\bar{B}\): the average betweenness value in current GCC. \(cS\): standard deviation of betweenness values in current GCC. \(d\) adaptive: \(k = 2^{(\text{int}(\log_2 k_l))}\), where \(k_l\) represents the number of nodes with betweenness \(\geq 0.5\) (e.g. if there are 9 nodes with betweenness \(\geq 0.5\), then \(k = 8\).)

| Category | Description | Parameters | Naming |
|----------|-------------|------------|--------|
| \(k = \text{constant}\) | Remove constant number of nodes | \(k\) in \([1, 2, 4, 8, 16]\) | e.g. 2 |
| \(k = \text{fraction of nodes}\) | Remove certain fraction of nodes | \(k\) in \([1\%, 5\%, 10\%, 20\%]\) | e.g. 20\% |
| \(k = \text{number of nodes with } B^a \geq \bar{B}^b + S^c\) | Remove all nodes with \(B^a \geq \bar{B}^b + S^c\) | \(\emptyset\) | AS |
| \(k = \text{adaptive}^d\) number of nodes with \(B^a \geq 0.5\) | Remove certain nodes with \(B^a \geq 0.5\) | \(\emptyset\) and adaptive\(^d\) in \([1, 2, 4, 8, 16]\) | 0.5 |

1. Identification of important nodes: In many cases, people are more concerned about the top nodes with high betweenness. We used three measures: Top-1\%-Hits, Top-5\%-Hits and Top-10\%-Hits.
2. Ranking sortedness: Compared with the exact ranking, the sortedness of approximated ranking can be described by inversion number.
3. Weighted coefficient: Considering the importance of top ranked nodes, we used Weightedtau to add the weight of exchanges between top ranked nodes.
4. Destructive to the network: During interactive computation, the size of GCC keeps decreasing as we keep removing nodes. A good method can identify nodes with high betweenness, which could have great impact on network connectivity, resulting in a quick dismantling process and a fast GCC reduction. We considered the number of nodes need to be removed to cut GCC into 10%.

In total, we devise six measures to evaluate the accuracy compared to the standard ranking (i.e. the ranking of nodes of exact computation with \(k = 1\)) as follows:

1. **Top-1\%-Hits**: The fraction of nodes correctly identified by approximate methods in Top-1\% nodes.
2. **Top-5\%-Hits**: The fraction of nodes correctly identified in Top-5\% nodes.
3. **Top-10\%-Hits**: The fraction of nodes correctly identified in Top-10\% nodes.
4. **Inversion**: Normalized inversion number of estimated ranking with exact ranking as the standard. After computing the inversion number, we normalized it and mapped it to \([0, 1]\): \(\text{Inversion} = (1 - \frac{2*\text{inv}}{N*(N-1)})\), where \(N\) is the number of nodes and \(\text{inv}\) is the exact inversion number.
5. **Weightedtau**: A node with rank \(a\) is mapped to weight \(1/(a+1)\) and an exchange between two nodes with rank \(a\) and \(b\) has weight \(1/(a+1)+1/(b+1)\). That is, top ranked nodes have higher weights, which increase the impact of the exchanges between important nodes.
6. **10\% GCC reduction (\(p_{10\%}\))**: it represents how many nodes the method requires to remove to dismantle the network until GCC < 10\%*\(N\). The normalized value \(p_{10\%}\) is mapped to \([0, 1]\) and 1 means the method which needs the minimum number of nodes to get 10\% GCC reduction.
TABLE II. Overview of generated random network types.

| ID | Name                     | Generator parameters                                                                 | Naming scheme       |
|----|--------------------------|--------------------------------------------------------------------------------------|---------------------|
| ER | Erdos-Renyi              | The number of nodes $n \in \{300, 700, 1000\}$,                                     | ER_n,DER            |
|    |                          | The probability for edge creation $p_{ER} \in \{0.015, 0.02, 0.025\}$.               |                     |
| BA | Barabási-Albert          | The number of nodes $n \in \{300, 700, 1000\}$,                                     | BA_n,m              |
|    |                          | The number of edges to attach from a new node to existing nodes $m \in \{2, 4, 6\}$. |                     |
| WS | Watts–Strogatz           | The number of nodes $n \in \{300, 700, 1000\}$,                                     | WS_n,kWS,pWS        |
|    |                          | The number of nearest neighbors that each node is joined with $k_{WS} \in \{3, 5, 7\}$.|                     |
|    |                          | The probability of rewiring each edge $p_{WS} \in \{0.2, 0.5, 0.8\}$.               |                     |

Runtime: We conduct experiments on the same computer with four i7-6500U cores (2.50GHz) and 16 GB RAM. We run each approximate method independently and recorded the exact runtime.

Trade-offs: Considering six measures of accuracy, we normalized the runtime and plotted it with normalized measures to see which method can offer a nice trade-off. In order to analyze the results on different networks, we computed the average normalized runtime and measure values. To sum up, we use six measures to evaluate accuracy and we also analyzed runtime and trade-offs. Besides, we set the naming schedule as approximation algorithm parameter $k$ (e.g. RAND2_64_2 represents using RAND2 algorithm with number of pivots = 64 and $k = 2$).

III. RESULTS

A. Networks in this study

First, we generated 9 ER (Erdos-Renyi) graphs, 9 BA (Barabási-Albert) graphs and 27 WS (Watts–Strogatz small-world) graphs with different sizes and parameters. Table II provides an overview of our random networks and generator parameters. Figure 4 visualizes four selected random networks. On these random graphs, we performed sensitivity analysis of Top-1-node-identification for three selected methods in order to select reasonable parameters (see below). In addition, we selected 48 real-world networks of different sizes and structures, covering a variety of domains, as obtained from http://networkrepository.com/networks.php:

- Social (4 networks): Networks showing the social friendships between people. Nodes are persons and edges represent their connections.
- Biological (5 networks): Networks showing the interactions between elements in biological systems.
- Brain (7 networks): Networks representing functional connectivity in brains. We chose different brain networks of mouse, macaque and fly.
- Ecology (2 networks): Networks showing the interactions between species.
- Economic (2 networks): Networks representing interactions between interconnected economic agents.
- Infrastructure (3 networks): Networks consisting of interlinks between fundamental facilities.
- Power (5 networks): Networks showing the transmission of electric power.
TABLE III. Overview of the real-world data sets.

| Category           | Range of Nodes   | Range of Edges   | Range of Density | Range of Maximum degree |
|--------------------|------------------|------------------|------------------|-------------------------|
| Social networks    | [889, 12645]     | [2914, 49132]    | [0.000615, 0.008053] | [102, 4800]             |
| Biological networks | [453, 3343]     | [1948, 6437]     | [0.001139, 0.019780] | [37, 523]               |
| Brain networks     | [29, 1770]       | [44, 16089]      | [0.003157, 0.712596] | [31, 927]               |
| Ecology networks   | [97, 128]        | [1446, 2075]     | [0.25529, 0.310567]  | [90, 110]               |
| Economic networks  | [257, 1258]      | [2375, 7513]     | [0.009502, 0.072197]  | [106, 206]              |
| Infrastructure networks | [332, 4941] | [2126, 15645]    | [0.00054, 0.038693]   | [19, 242]               |
| Power networks     | [494, 5300]      | [586, 8271]      | [0.000589, 0.040812]  | [9, 17]                 |
| Road networks      | [1039, 2640]     | [1305, 3302]     | [0.000948, 0.0242]    | [5, 10]                 |
| Technological      | [2113, 10680]    | [6632, 24316]    | [0.000426, 0.002972]  | [109, 205]              |
| Web networks       | [643, 12305]     | [2280, 47606]    | [0.000258, 0.011046]  | [59, 199]               |
| Email networks     | [143, 1133]      | [623, 5451]      | [0.0085, 0.061361]    | [42, 71]                |
| Retweet networks   | [2280, 9631]     | [2464, 10314]    | [0.000211, 0.000948]  | [267, 7655]             |
| Cheminformatics networks | [123, 125] | [139, 141]       | [0.018194, 0.018526]  | [5]                     |

FIG. 5. Top-1-Hits of different static betweenness approximation techniques on random networks.

- **Road** (2 networks): Networks representing roads connectivity between intersections.
- **Technological** (2 networks): Networks consisting of the interlinks between technology systems.
- **Web** (5 networks): Networks representing the hyperlinks between pages of the World Wide Web.
- **Email** (2 networks): Networks showing mail contacts between two addresses.
- **Retweet** (7 networks): Networks describing retweeting relationships on the Twitter.
- **Cheminformatics** (2 networks): Networks reflecting the chemical interactions of materials.

Table III shows an overview of our 48 real-world data sets, including network properties.

B. Sensitivity analysis / parameter selection

In order to selected reasonable parameters of approximation methods, we evaluated the quality of identification of the Top-1 node with each selected method by computing static betweenness with each method on our generated random networks. Figure 5 reports the fraction of networks on which each competitor can correctly identify the Top-1 node.

**RAND2**: Figure 5 shows the results of identifying the Top-1 node on random networks regarding different number of sampled pivots. It can be seen that the quality (measured as ratio of correctly identified nodes) increases with the number of pivots and sampling with 512 pivots is the best. **RAND2_64** can be chosen as a trade-off one and it correctly identifies over 70% WS networks and saves much time.
The quality of RAND2 and KPATH with $k = 1$ is outstanding. Figure 6 can save runtime compared to RAND2 and KPATH. Considering the prohibitive computational costs on larger networks, we selected $512$, RK with different $k$. Table IV shows the measure values and runtime $64$, RK is not good. We computed the average measure values $8$ is reasonable among all selected settings. RK offers the highest accuracy in general. We computed the average measure values for further analysis on $48$ real-world networks.

### C. Accuracy

Since the computation on real-world networks is expensive, we analyzed results on generated random graphs first, in order to select competitors and conduct further experiments on real-world networks. Figure 6 presents the average measure values of $66$ competitors. We can see that RAND2 offers the highest accuracy in general.

Figure 7 presents the distribution of $10\%$ GCC reduction of $66$ competitors. We can see that on $10\%$ GCC reduction, RAND2, RK, RK and KPATH, KPATH, KPATH is reasonable among all selected settings.

With these results, we selected RAND2, RAND2, RK, RK, RK, RK, KPATH and KPATH for further analysis in the remaining part of the study.

**RK**: As Figure 5 indicates, it can get the best quality when we set $\epsilon = 0.07$. However, it only identified $60\%$ of all random graphs. As RK with $\epsilon = 0.2$ and $0.3$ cannot identify Top-1 node in all ER networks, we chose $\epsilon = 0.07$ and $0.1$.

**KPATH**: The results of KPATH are shown in Figure 5. The quality is the worst compared to RAND2 and RK. KPATH, KPATH, KPATH, KPATH is reasonable among all selected settings.

With these results, we selected RAND2, RAND2, RK, RK, RK, RK, KPATH and KPATH for further analysis in the remaining part of the study.

**Real-world networks**: We run experiments on $48$ real-world networks and computed six measure values of accuracy. Figure 8 presents the distribution of measure values. We can see that RAND2 with $k = 1$ is outstanding on all measures. Besides, when setting constant $k$ values, the quality becomes worse as we increase the $k$ value. On measure $p_{10\%}$ (GCC $10\%$ reduction), it is clear that the quality becomes worse from $k = 1\%$ to $k = 20\%$. Besides, RAND2 and RK with removing certain nodes with $B \geq 0.5$ can also offer good accuracy. Compared to RAND2 and RK, the quality of KPATH is not good. We computed the average measure values

![FIG. 6. Average measure values of 66 competitors on 45 random networks.](image1)

![FIG. 7. Distribution of 10% GCC reduction of 66 competitors on 45 random networks.](image2)
TABLE IV. Results on ER networks with 300 nodes with generator parameter \( p = 0.02 \), using six approximation algorithms and setting \( k = 1 \).

| Measure       | RAND2_512 | RAND2_64 | RK_0.07_0.1 | RK_0.10_0.1 | KPATH_0.2_8 | KPATH_0.2_4 |
|---------------|-----------|-----------|-------------|-------------|-------------|-------------|
| Runtime (s)   | 39.7      | 22.9      | 28.5        | 23.7        | 8.1         | 4.2         |
| Top-1%-Hits   | 0.75      | 0.75      | 0.75        | 0.25        | 0.75        | 0.75        |
| Top-5%-Hits   | 0.75      | 0.75      | 0.81        | 0.69        | 0.88        | 0.81        |
| Top-10%-Hits  | 0.90      | 0.87      | 0.81        | 0.81        | 0.77        | 0.67        |
| Weightedtau   | 0.01      | 0.12      | 0.08        | 0.17        | 0.02        | 0.01        |
| Inversion     | 0.78      | 0.81      | 0.82        | 0.80        | 0.84        | 0.84        |
| \( p_{10\%} \) | 0.97      | 0.97      | 0.94        | 0.94        | 0.83        | 0.69        |

FIG. 8. Distribution of measure values on 48 real-world networks.

on 48 real-world networks and the results are shown in Figure 9: RAND2_64_1, RK_0.10_0.1_1, RAND2_64_0.5 and RK_0.10_0.1_0.5 are good.

D. Runtime

The runtime of computing interactive betweenness depends on the size of the network, choices of \( k \) and selected approximation algorithms. We analyzed the runtime regarding different \( k \) values with the same approximation method. Besides, we evaluated the runtime of different approximation methods with the same \( k \) setting.

Runtime regarding different \( k \) settings: Figure 10 plots the runtime (in seconds) of RAND2_64, RK_0.10_0.1 and KPATH_0.2_4 with the same \( k \) setting (i.e., \( k = \) certain nodes with \( B \geq 0.5 \)) on different real-world networks with y-axis = runtime in seconds and x-axis = NlogN where \( N \) is the number of nodes in the network. We can see that Figure 10 shows that the time complexity is around \( O(N \log N) \) for these sparse real-world networks. Note that, for dense networks, the runtime will be nearly \( O(N^2) \) theoretically. Moreover, the runtime of RAND2_64 is the highest. While, KPATH_0.2_4 is the fastest among three approximation methods but it does not offer a nice quality.

Runtime regarding different approximation methods: Figure 11 shows the runtime of RK_0.10_0.1 with \( k \) from 1 to 16. We can see that the runtime increases as \( k \) decreases. If we choose smaller \( k \), then fewer nodes are removed in each iteration, resulting in larger number of iterations and computational costs. Besides, doubling the
In this subsection, we present the speedup of interactive betweenness approximation compared to the standard BETWI (exact interactive betweenness computation). Based on our experimental results of BETWI and other approximation methods conducted on the same computer, we computed the speedups on three ER networks with different sizes (N = 300, 400, 500, 600, 700, 800, 900 and 1000) but the same generator parameter p = 0.02. Similar approximations methods conducted on the same computer, we computed the speedups on three ER networks with different sizes (N = 300, 400, 500, 600, 700, 800, 900 and 1000) but the same generator parameter p = 0.02. Similar

E. Speedup

In this subsection, we present the speedup of interactive betweenness approximation compared to the standard BETWI (exact interactive betweenness computation). Based on our experimental results of BETWI and other approximation methods conducted on the same computer, we computed the speedups on three ER networks with different sizes (N = 300, 400, 500, 600, 700, 800, 900 and 1000) but the same generator parameter p = 0.02. Similar
to our analysis of runtime, our evaluation on the speedups of interactive betweenness approximation sheds light on two aspects, the speedups of different betweenness approximation algorithms and the speedups with increasing $k$. Figure 12(left) shows the speedups of RAND2$_{512}$, RAND2$_{64}$, RK$_{0.07}_0.1$, RK$_{0.10}_0.1$, KPATH$_{0.2}_4$ and KPATH$_{0.2}_8$ with the same $k$ setting. We can see that the speedup increases as the network becomes larger. As a fast algorithm, KPATH offers great speedups compared to RK and RAND2. Figure 12(right) presents the speedups with different $k$ settings. Removing one node from GCC in each iteration induces low speedups. While, doubling the $k$ value approximately doubles the speedup.

**F. Trade-offs**

From the results on quality and runtime, some competitors (e.g. RAND2$_{64}$0.5) gets high quality but need hours on the largest network. Several methods (e.g. KPATH$_{0.2}_4$ with $k = 16$) is quite fast but the quality is bad. In this subsection we focus on the trade-offs of these selected competitors.

**Trade-offs on specific networks:** Figure 13 presents trade-offs between quality (i.e., the values of six measures of accuracy) and speed (exact runtime). We used 3 colors to distinguish 3 approximation methods and 3 markers to label 3 typical $k$ settings, including a faster one $k = 16$, a slowest one $k = 1$ and $k = 4$ as a trade-off. We can see that the RK$_{0.10}_0.1$ with $k = 4$ gets nice trade-offs on Top-1%-Hits, taking no more than 25% runtime to get high accuracy compared to the maximum runtime. While, when we consider Inversion measure, KPATH$_{0.2}_4$ with $k = 1$ is good. In addition, RAND2$_{64}$ with $k = 4$ also provides pleasurable trade-offs on both Top-1%-Hits and Weightedtau.

**Average trade-offs:** As there are orders of magnitude deviation between runtime on different networks, to analyze the trade-offs on 48 real-world networks, we normalized the runtime on each network with the slowest = 1 and the fastest = 0 and then we computed the average normalized runtime on 48 networks. Besides, we further normalized the measure values to map these values into $[0, 1]$ on each network and computed the average normalized measure values. Figure 14 shows the results. We used the same labels as Figure 13 and add legends of these competitors with average normalized runtime $\leq 0.5$ and average normalized measure values $\geq 0.6$. We can see that setting $k$ in $[2, 4, 1\%, 0.5]$ can make good trade-offs with specific approximation methods.

**IV. CONCLUSIONS**

Betweenness centrality is a widely used measure of node importance, which counts the number of shortest paths a node appears in a network. However, if one node in the network is being attacked or loses its functionality, the betweenness values of other nodes will change. That is, all betweenness values need to be recomputed, in order to update the actual node importance. Recent research suggests that, on network dismantling problem, interactively removing one node with the highest betweenness outperforms removing nodes based on ranking obtained by one time
betweenness computation. However, the interactive betweenness computation requires static betweenness recomputation on current GCC after each node removal and it is significantly more computationally expensive (an order of
magnitude) compared to static approaches.

In this paper, we systematically investigate approximation of interactive betweenness centrality. We proposed a framework for interactive betweenness estimation with k-batch removal. Our framework consists of a set of static betweenness approximation algorithms with various parameter settings for identifying top nodes with high betweenness and selections of how many nodes to be removed in each iteration. In other words, we not only analyzed the performance of removing one top node but also evaluated the removal of a batch of nodes. As the computation of interactive betweenness is more expensive than the computation of static betweenness, we focus on choosing approximation methods with parameter settings and k values (the number of nodes to be removed in each iteration) which can offer high quality and also a nice trade-off between accuracy and speed. To ensure that our data sets cover different network structures, we generated 45 random networks, including ER, WS and BA networks, and selected 48 real-world networks with distinct sizes from different fields. We devised six measures to evaluate accuracy with considering the identification of important node, the similarity of rankings and the effects on GCC reduction.

To make preliminary selections of suitable parameter settings, we conducted sensitivity analysis of static betweenness approximation algorithms and evaluated the quality of identification of Top-1 node on random networks. We selected six approximation methods, consisting of RAND2, RAND2_512, RK_0.07, RK_0.10, KPATH_0.2, and KPATH_0.2_8. As for k settings, based on the results of 50% GCC reduction, we found that many networks can be dismantled with a small fraction of nodes and we choose 11 different k setting (k in [1, 2, 4, 8, 16, 1%, 5%, 10%, 20%, AS, 0.5]). We run tests with 66 competitors (six approximation algorithms with 11 k settings) on random networks to further select competitors. Based on the results on random networks, we chose RAND2, RK_0.10, and KPATH_0.2_4 with 11 k settings and conducted experiments on larger real-world networks. We found that RAND2.64, RAND2.64.5, RK.010,0.1, and RK.010,0.1,0.5 offer high accuracy. Besides, we analyzed the runtime regarding different approximation algorithms k settings. Our analysis on different approximation methods with the same k reveals that RAND2.64 is the slowest and KPATH.0.2 is the fastest competitor. Moreover, we also found that doubling k values will get 50% runtime reduction for k ≥ 2 and the runtime reaches its upper bound with k = 1 (shown by Figure 11). Our analysis on trade-offs indicates that RAND2_64 and RK_0.10 with k = 2, 4, 1% and 0.5 can offer nice trade-offs between accuracy and speed.

In synthesis, we have proposed a novel framework for interactive betweenness approximation. We systematically evaluated the selections of approximation algorithms with various parameter settings and the choices of different batch removals from three aspects: accuracy, runtime and trade-offs between them. Our work contributes to the analysis of complex network phenomena, with a particular focus on obtaining scalable techniques. Future work could investigate the interactive approximate computation of other network centrality measures.

ACKNOWLEDGEMENTS

This study is supported by the Research Fund from National Natural Science Foundation of China (Grants No. 61861136005, No. 61851110763, No. 71731001).

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

DATA AVAILABILITY

All networks used on this study are available from the public repository http://networkrepository.com/networks.php.

V. REFERENCES

[1] Albert R, Albert I and Nakarado G L 2004 Physical Review E 69 025103
[2] Cuadra L, Salcedo-Sanz S, Del Ser J, Jiménez-Fernández S and Geem Z W 2015 Energies 8 9211–9265
A. STATIC BETWEENNESS ESTIMATION TECHNIQUES

Pivots sampling: [25] introduced RAND1 for betweenness approximation. RAND1 samples a subset of source nodes uniformly at random and computes the estimated betweenness of all nodes with scaling it up by $N/S$, where $|S|$ is the number of sampled source nodes. [26] proposed GSIZE algorithm which determines the number of sample pivots by graph size. GSIZE utilizes an adaptive sampling technique which was introduced by [27]. Given a node $v$, GSIZE keeps sampling pivot $s$ until $\sum \delta_s(v) > 5 \cdot N$. [28] proposed RAND2 based on random sampling to approximate static betweenness values of all nodes. RAND2 modifies RAND1 by scaling it up with a linear function. RAND2 decreases the contribution of nodes closed to the source nodes and can solve the overestimation problem of RAND1.

Node pairs sampling: [29] proposed a fully dynamic algorithm (DA) for computing estimated betweenness. DA keeps track of the old shortest paths and substitutes them only when they are necessary. [30] proposed RK which samples pairs of nodes instead of conducting BFS from sampled source nodes. RK is an $(\varepsilon, \delta)\text{-approximation}$: given the allowed additive error $\varepsilon$ with probability $1 - \delta$, RK guarantees that the error of each node is less than $\varepsilon$ with probability at least $1 - \delta$. RK determines the sample size by VC-dimension (Vapnik-Chervonenkis dimension) introduced by [31] instead of the network size. [32] presented an $(\varepsilon, \delta)\text{-approximation}$ method ABRA. ABRA uses progressive sampling and sets the stop condition by utilizing Rademacher Averages proposed by [33] and pseudodimension introduced by [34] in statistical learning field.

Bounded BFS: [35] found that the betweenness of node $v$ in its EGO network is related to the exact betweenness of $v$ in the network. The ego network of $v$ is composed of node itself, all the neighbors of $v$ and the edges that connect those nodes. [35] used neighbors with distance $\leq 2$ in their EGO approximation algorithm. In other words, EGO bounds the BFS with 2 hops from source nodes. [36] presented KPATH method which computes betweenness centrality values based on $k$-centrality measures. [36] assumed that these nodes distant from each other do not contribute to the betweenness values. Compared to [35], the BFS of KPATH is bounded by $k$ hops from source nodes and these nodes with distance $> k$ from the source node are not considered. [37] introduced an adaptive algorithm KADABRA, which can approximate betweenness of all nodes or just compute the Top-k nodes. KADABRA uses a balanced bidirectional BFS to sample shortest paths. Instead of conducting a full BFS from $s$ to $t$, KADABRA performs a BFS from $s$ and a BFS from $t$ at the same time until such two BFSs touch each other.

As mentioned above, we divided approximation algorithms into three classifications: pivots sampling, node pairs sampling and bounded BFS. We selected one method with a nice trade-off between runtime and quality from each classification. For pivots sampling methods, we chose RAND2 as it offers an outstanding accuracy with a good trade-off. From an experimental perspective, the results of [38] and [39] both show that RAND2 outperforms other methods in tested networks. From a theoretical perspective, with linear scaling, RAND2 can handle the overestimation problem of RAND1. Thus, RAND2 can be selected as a representative of the methods based on pivots sampling. However, the performance of RAND2 is determined by sample size. As RAND2 needs $|S|$ (the number of sampled pivots) iterations of BFS, the time complexity of static RAND2 is $O(|S| \cdot E)$. On the one hand, if we sample few pivots, we can not identify the Top-1 node (the node with the highest betweenness) well. On the other hand, if we sample too many pivots, we will do redundant calculations. As [28] suggests, we just selected constant sample sizes $|S| \in \{8, 16, 32, 64, 128, 256, 512\}$. We selected RK proposed by [30] among node pairs sampling methods. The results on Top-1%-Hits in the benchmark provided by [38] indicate that RK is a better choice for identifying vital nodes. As an $(\varepsilon, \delta)\text{-approximation}$ method, the $\varepsilon$ can greatly affect the speed and quality by determining the sample size [40]:

$$r = \frac{1}{2\varepsilon^2} \left( \left\lfloor \log_2 (VD(G) - 2) \right\rfloor + 1 + \ln \frac{1}{\delta} \right)$$

(3)

where $VD(G)$ is estimated vertex-diameter of the network as the computation of exact vertex-diameter is quite expensive. Since we focus on identifying the Top-1 node for interactive approximation, we can set $\varepsilon$ higher than 0.01 (default). We evaluated the performance of RK with $\varepsilon \in \{0.07, 0.1, 0.2, 0.3\}$. As for $\delta$, we set it to be 0.1 (default). In addition, we chose KPATH introduced by [36] as a typical one among the methods based on bounded BFS. KPATH approximates static betweenness centrality values with using $k$-centrality measures. KPATH assumes that nodes distant from $v$ offer zero dependencies. KPATH stops the BFS until reaching $k$ hops. Therefore, only pair dependencies of two nodes with distance $\leq k$ can contribute to the betweenness values. KPATH determines sample size by parameter $\alpha$: the number of samples is proportional to $N^{1-2*\alpha}$, where $N$ is the number of nodes in the network. To distinguish $k$ in KPATH from our $k$ batch removal, we name $k$ in KPATH $k_{\text{KPATH}}$. We set $k_{\text{KPATH}} \in \{4, 8\}$. Besides, considering the $\alpha$ values, we set it to be 0.0, 0.2 and 0.4 to make comprehensive comparison. For three
TABLE V. Overview on the selected methods of betweenness approximation.

| Method   | Parameters                  | Description                                      |
|----------|------------------------------|--------------------------------------------------|
| Betw     | $\emptyset$                 | Brandes’ algorithm for computing exact betweenness. |
| RAND2    | $|S| \in \{8, 16, 32, 64, 128, 256, 512\}$ | Sampling pivots uniformly at random.             |
| RK       | $\epsilon \in \{0.07, 0.1, 0.2, 0.3\}$, $\delta = 0.1$ | Using $VD(G)$ and sampling node pairs.           |
| KPATH    | $\alpha \in \{0.0, 0.2, 0.4\}$, $k_{KPATH} \in \{4, 8\}$ | Bounded BFS within $k_{KPATH}$ hops              |

selected methods with its parameter settings, we choose the naming scheme as: method.parameter (e.g., KPATH$_{0.2, 4}$ is KPATH method with $\alpha = 0.2$ and $k_{KPATH} = 4$). Table V presents an overview of our selected methods.