2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

# Local Detection of Critical Nodes in Active Graphs

M. Yusuf Özkaya
Georgia Institute of Technology
Atlanta, GA
myozka@gatech.edu

Ahmet Erdem Sarıyüce
University at Buffalo
Buffalo, NY
erdem@buffalo.edu

Ali Pınar
Sandia National Laboratories
Livermore, CA
apinar@sandia.gov

Ümit V. Çatalyürek
Georgia Institute of Technology
Atlanta, GA
umit@gatech.edu

***Abstract*—The identification of critical nodes in a graph is a fundamental task in network analysis. Centrality measures are commonly used for this purpose. These methods rely on two assumptions that restrict their applicability. First, they only depend on the topology of the network and do not consider the activity over the network. Second, they assume the entire network is available. However, in many applications, it is the underlying activity of the network such as interactions and communications that makes a node critical, and it is hard to collect the entire network topology, when the network is vast and autonomous.**

**We propose a new measure, *Active Betweenness Cardinality*, where the importance of the nodes are based not on the static structure, but the active utilization of the network. We show how this metric can be computed efficiently by only local information for a given node and how we can locate the critical nodes by using only a few nodes. We also show how this metric can be used to monitor a network and identify node failures. We evaluate our metric and algorithms on real-world networks and show the effectiveness of the proposed methods.**

## I. Introduction

Identifying critical elements of a network is an important task in infrastructure security and social network analysis. The critical elements are those, whose presence are essential for a network to maintain its functionality at or near its maximum. Subsequently, the network science community have proposed many metrics to quantify the criticality of network elements and algorithms to compute these metrics efficiently.

A major thrust in this area is *centrality*, which assigns a number to each node as a measure for its importance. For details on centrality, we refer the reader to [1]. Centrality measures are valuable, but they all rely on two assumptions, which restrict their effectiveness and applicability (1) they depend only on the topology of a graph and not consider the dynamics on the graph or how the graph is being utilized, and (2) entire graph is available. First, we cannot assume interactions between all pairs of nodes on a large graph to be at the same level or frequency. This may be due to lack of interest (e.g., not every user visits every web page) or some nodes provide identical services and users only interact with the nearest such server (one goes to a close-by hospital, not necessarily all the hospitals). It should be noted that centrality measures were initiated by the social science studies on much smaller networks, where neither of the these two limitations apply. But one should be careful before applying the same ideas to current datasets, where the graphs are much larger and contain nodes with identical functionalities. Secondly, many networks such as the Internet are vast, distributed, and autonomous. Moreover, both their topology and the dynamics on them keep evolving. Determining the exact topology or even predicting their topological features are far from trivial.

IEEE/ACM ASONAM 2018, August 28-31, 2018, Barcelona, Spain
978-1-5386-6051-5/18/$31.00 

We propose a new approach for centrality that considers the network dynamics and does not require having access to the entire network. We call our approach the *Active Betweenness Cardinality (ABC)* model. The metric for our model is the number of distinct pair of nodes that communicate through a node within a specified period. In our data access model, we know the source and destination for any interaction through a node. We develop methods to compute the ABC value of any node with **only local information**, i.e., knowing nothing about the rest of the graph. We estimate the ABC values efficiently (both in terms of runtime and memory) and accurately, using a sublinear algorithm. We also show how to locate the nodes with high ABC values and how to utilize this metric to identify presence and the location of changes in a network. We claim that this approach provides a critical capability for analysis for vast distributed networks.

Let $G = (V, E)$ be a simple undirected, unweighted graph with $n$ nodes and $m$ edges. We assume any pair of nodes can exchange messages and any interaction can be repeated. We define $G$ as an **active network**. If there is an interaction between nodes $u$ and $v$, it happens via a path $p$ between $u$ and $v$, and all the communication thereafter also happens via the same path $p$. For consistency with routing algorithms [2], we restrict $p$ to be one of the shortest paths between $u$ and $v$. We define the number of **unique** interactions that are transmitted by a node $i$ as the *cardinality* of $i$. We want to detect the nodes with large cardinality by using only the local information.

*1) Cardinality Estimation on Data Streams:* An important part of our problem is estimating the cardinality of a data stream where we observe the elements of a set that can occur repeatedly. Formally, given a stream $x_0, x_1, \ldots x_K$ with repetitions, where $x_i \in S$ for $i = 0 \ldots K$, we have a dual objective function: We want to estimate $|S|$, the cardinality of the set $S$, as accurately as possible while using minimal storage. For the purposes of this paper, the elements of the stream will be the interacting pairs, and each entry in the system will be one message exchange.

The difficulty of this problem lies in handling the repetitions. A trivial solution is to maintain a hash table to keep track of the previously observed elements. This gives an exact solution, but associated memory requirement will be linear in the size of the set $S$, which is impractical for many real- world scenarios. The algorithmic challenge is in drastically reducing storage, while maintaining accuracy. Using randomization for

the cardinality estimation, initiated by Flajolet and Martin [3], is helpful to address this challenge. For a thorough survey of this literature, we refer the readers to [4]. In our experiments we used the HyperLogLog (HLL) method with parameter $b = 12$ bits, and thus, $m = 2^b = 2^{12}$ buckets, though our methods do not depend on any particular cardinality estimation method, we selected HLL for its well-accepted success.

*2) Centrality Measures:* The centrality metrics play an important role in network and graph analysis since they are related with several concepts such as reachability, importance, influence, and power [5]–[8]. Betweenness and closeness centralities (BC and CC) are two such metrics.

A lot has been done on the centrality measures to improve their scalability with more efficient algorithm design [9], approximation schemes [10], parallel algorithms [1], [8], [11]–[13], and using sublinear memory in the size of the graph [14], [15]. However, all of those approaches focus on the graph topology information to infer the centrality, i.e., communications between each pair of nodes are assumed to be same and non-repetitive, and the entire graph should be known to highlight the most central nodes. We propose local algorithms that can find the most critical nodes. Our focus is on active networks where there can be non-uniform and repetitive communications between nodes, which, to the best of our knowledge, is not studied before.

## II. Active Betweenness Cardinality

### A. A New Metric for Node Criticality on Active Networks

We introduce *Active Betweenness Cardinality* (ABC) as a new metric that measures node criticality not only based on topology, but on the activity in the network. ABC also has the advantage that it can be computed based on only local information without knowing the entire graph. Hence, our metric relieves us from both assumptions explained in Sec I. ABC of node $v$ for time period $[t_b, t_e]$, $ABC(v, t_b, t_e)$ is the number of distinct source-destination $\langle s, d\rangle$ tuples for an intermediate node $v$, which we will also call as *transaction*. We will only refer to the ABC value of a vertex, when the time interval is clear from the context.

In the Internet context, a node $s$, connecting to the service of a node $d$, may require multiple packages going back and forth over $v$. The failure of this communication path causes the user to lose access to one *service*. Thus, this behavior can be modeled better with distinct tuple count than total count of transactions.

### B. Modeling Network Activity

Table I shows the data sets used in our experiments. As for modeling network activities, there are 3 main aspects to model: communication pathways, node activities, and time window. We follow the model explained in detail in [16]. To summarize, communication between a pair of nodes follows a randomly selected shortest path throughout the experiment unless there is a change in the graph structure. We model the communication behaviors between node pairs by assigning certain send/receive frequency levels to nodes. The probability of a communication from node $u$ to node $v$ is proportional to the product of send frequency level of node $u$ and the receive frequency level of node $v$. The rest of the paper presents the results using the Gaussian distribution. And finally, we have selected a sufficiently long time window to observe the numbers after convergence to avoid additional variables in the presentation of the algorithm. For practical purposes, we define *interval* as the number of communications over the graph instead of actual time units since in essence, it achieves the same purpose.

TABLE I: Graphs used in the experiments

| Graph | #Nodes | #Edges | Degree | | |
|---|---|---|---|---|---|
| | | | min | avg | max |
| World Airports [17] | 2,939 | 30,501 | 1 | 10.37 | 237 |
| AS-733 [18] | 6,474 | 12,572 | 1 | 3.88 | 1,458 |
| Oregon-01 [18] | 10,670 | 22,002 | 0 | 4.12 | 2,312 |

### C. Computing ABC Accurately and Efficiently

First, we show how the ABC metric can be computed efficiently, using only local information. We use HLL algorithm to compute the cardinalities.

We start with verifying the HLL accuracy on our experiments. HLL algorithm consistently provides good estimates, always within $\pm 2\%$ range. Average Pearson Correlation for *Exact* and *HLL estimated* cardinalities was 0.99994. We refer the reader to the extended version [16] of this work for more detailed explanation and experiments.

### D. Finding Nodes with Highest ABC Scores

Here, we propose a set of heuristic search strategies to identify nodes with the highest ABC scores in a distributed fashion, starting from a handful of nodes, under the assumption that we do not know about the presence of all the nodes in the graph. Then, we empirically show their effectiveness.

*1) Search Strategies:* Our approaches are based on choosing the neighbor of the current node most promising to have a higher ABC value. We avoid revisiting a previously seen node. Starting from $C$ *seed* nodes, the strategies select $C$ new nodes to continue the search. We call each iteration a *hop*.

We show three approaches to find important nodes:

*Jump to Best Neighbor (BN)*: For each neighbor of a seed node, we maintain an HLL estimator (ABC value) that counts the distinct pairs that communicated through that neighbor/edge. Intuitively, if we have an important neighbor, it would acquire most of our communication output, like a communication hub. Following this, we select the neighbor with the highest cardinality on the edge that connects the current node to that neighbor as one of the next *seed* nodes.

*Jump with Random Walk (RW)*: We implemented a random-walk-like approach where the neighbors have probabilities proportional to their corresponding ABC values on the edges. For example, a node $v$ has neighbors $w$, $x$, $y$, and $z$ and the ABC values on the edges are $\{34, 16, 36, 12\}$, respectively. With *BN* strategy, we would jump to $y$ and we might miss the chance to see $w$. However, we can avoid this by treating the ABC values as probabilities and randomly pick a neighbor with respect to ABC values.

*Biggest Collective Neighbor Selection (BCN)*: *RW* and *BN* lets each seed node select one node. Alternatively, *BCN* lets them collectively decide which nodes should be in the next

seed set. This will speed up the convergence in cases where one of the subset nodes have all important neighbors but all the others are not as important. For example, consider a scenario where the seed set is $\{w, x, y, z\}$, and all the neighbors of node $w$ has ABC values an order of magnitude bigger than that of $x$, $y$, and $z$. In this case, it would make much more sense to select the new nodes from the neighbors of node $w$ instead of picking one neighbor from each seed node. More generally expressed, we select the top $C$ non-observed neighbors from the union of neighbors of current seed set.

*2) Quality Metric for Search Strategies:* In this section we define the quality metrics for the search strategies. First metric is one-to-one correspondence of the critical nodes found in the *ground truth*. Second, the sum total of ABC values found compared to the *ground truth*. What constitutes a *ground truth (baseline)* depends on the activity of the network, therefore, we need to have a ground truth for that specific configuration. The activity on the network follows a probability distribution, thus, it fluctuates around a mean value. We generate samples and use the mean of these samples as the baseline, i.e., we generate a baseline by averaging the ABC value of each node over 400 samples with the same configuration.

*Top-$K$ Found Matches*: For all algorithms, we compare the top $K$ nodes found with the top $K$ in the baseline. If the node found by the algorithm is actually among the top $K$ of the baseline, then we count this as a *hit*. So, perfect result would be to find $K$ hits out of $K$. We report the average of 20 runs for the experiment.

*Total Cardinality of Top-$K$ Found*: If the cardinalities of the most critical nodes are not distinguishably different, comparing top-$K$ vertices will not be fair. For such cases we compare the the sum of the total cardinality of top-$K$ found against the sum of the total cardinality of top-$K$ of baseline.

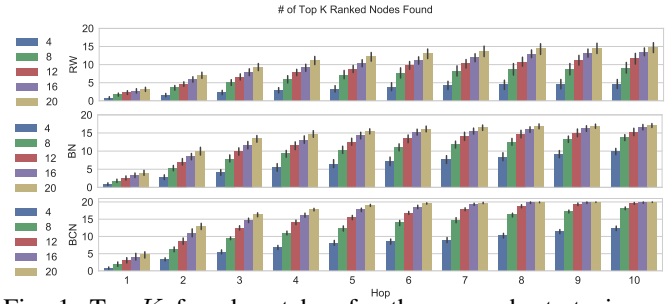


Fig. 1: Top-$K$ found matches for three search strategies on Airports graph (Top to bottom: RW, BN, BCN).

*3) Experimental Results:* Figure 1 shows the comparison of the search strategies in terms of top-$K$ nodes found for the Airports graph with $K = 20$. Each bar represents a different seed set size, i.e., $C = \{4, 8, 12, 16, 20\}$. As expected, as the seed set size $C$ increases, so the quality of the results. Our search strategies require small number of hops to stabilize, and they succeed to identify large portion of the critical nodes with RW and BN variants, and all top-$K$ nodes with *BCN* even with modest sizes of $C$, where $C \approxeq K$. The figure also depicts that the quality of the approaches increase from top to bottom, having *BCN* the best and *RW* slightly worse results. It also shows the results tend to converge in less than 6 hops for all 3 of the graphs and reach a reasonable 75% with seed set sizes $\{12, 16, 20\}$. Note that this means it reaches this conclusion after checking only up to $6 \cdot |C|$ nodes (72, 96, and 120, respectively, for the seed set sizes) out of all nodes. Finally, *BCN* also converges quicker than the others.

Figure 2 shows a comparison of the search strategies using our second metric, total cardinality of top-$K$ ($K = 20$) found, where the baseline is displayed as blue solid line on the charts. We show the results for Oregon graph. The total cardinality found by each algorithm gets very close to the one identified in baseline, implying that the nodes selected by *BN* and *RW* in Figure 1 from top-$K$ have similar ABC values to those missed from the baseline.

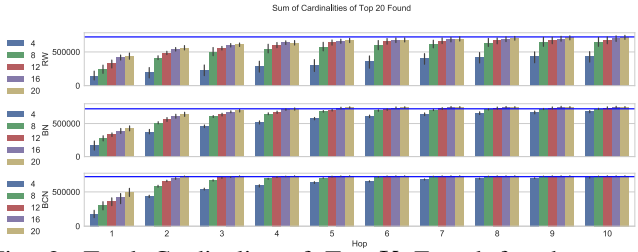


Fig. 2: Total Cardinality of Top-$K$ Found for the search strategies on Oregon Graph (Top to bottom: RW, BN, BCN).

## III. Detecting Changes in Network

We present how the ABC values can be used to monitor a network and detect significant changes in its topology. We first show how to detect the failure of a critical node by using a small number of sensors, then we discuss how to locate those nodes. *Sensor* is a node that computes its ABC value for a predefined time window and reports this information for further analysis. We restrict ourselves to detecting failures of nodes with high ABC scores (e.g., top 200), as failures of such nodes make an overall impact on the functionality of the network. We ask if this change can be captured by monitoring only ABC values of a limited number of sensors.

The critical observation is that when the underlying network is stable, the ABC values are steady. However, the same ABC scores are sensitive to changes in the network, as we will show.

### A. Detecting and Locating the Failure of Critical Nodes

Our initial experiments showed that the supervised learning with SVM (rbf kernel) have near 100% accuracy in sensing the removal of a critical node subset. Figure 3 motivated our

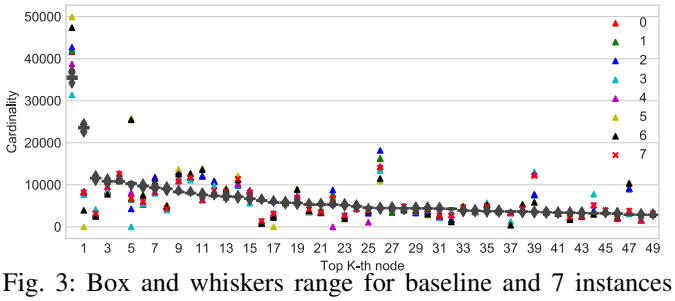


Fig. 3: Box and whiskers range for baseline and 7 instances of critical node removals. The box corresponds to the first and third quartiles. Whiskers show the range for $\pm 1.5 \times IQR$ (the difference between the third and first quantile).

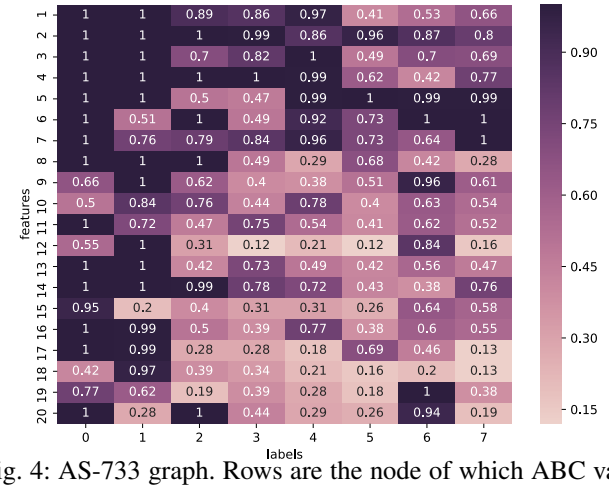


Fig. 4: AS-733 graph. Rows are the node of which ABC value is used as classifier. Columns are the classes(labels), that is, top 7 nodes being disconnected from the network one-by-one. Observing top i-th node's (y-axis) ABC value helps classify the top i-th node's (x-axis) removal. Class (label) 0 is the graph when there is no change.

work to go a step further. It shows the changes in top 50 ABC values in the Airports graph for intervals of size 200,000 transactions where the top 7 nodes are failed. We plot these results on the baseline (box and whiskers). Label 0, shows *typical* case, and the labels 1-7 correspond to failure of the node with corresponding ABC rank. We observe each failure pushes at least one ABC value significantly out of range.

The problem is trivial when we have sensors on all nodes, we would look at the incoming or the lack of incoming data from each node. However, we want to infer beyond what we can observe to reduce the amount of work and cost. The optimal is to have one sensor to detect all. The smaller number of sensors we need, the better. To see what is the minimum we need, we experiment what we can identify by each sensor alone. We consider each node in the top $M$ individually and use it as the only information source. That is, we only look at the ABC value of *one* node (called as *feature*) for the classification algorithm. The experiment dataset consists of 200 (10 training and 190 test) instances per label each run.

We store the accuracy achieved by each *feature* for *label*s (node failures) in a feature - label matrix. Figure 4 shows the results for AS-733. Noting the assumption there is no other change in the graph, we show that the classification algorithm successfully ($\geq \%90$ accuracy) works in most cases. We want the Minimum Set Cover on feature - label matrix, where we try to cover all the labels (failure cases) with the minimal number of features (sensors) selected. For example, in Figure 4 we can select $4th$ (or $2nd$) and $5th$ as our features to successfully cover all the labels with 2 nodes. We confirmed our findings by repeating this experiment using combination of multiple nodes. We refer to [16] for further experimental results.

### B. Effect of Noise in Experiments

Not every node on the communication graph behaves the same all the time. We achieve this in our experiments by incorporating noise factor to the communication probabilities. While each node initiates communication with others proportional to its send frequency, and receives communication proportional to its receiver frequency, these should be varying from time to time (between intervals) so that we do not assume that every node behaves exactly the same all the time. To test the robustness, we have added a multiplicative noise factor uniformly random in the $\times 0.8 - \times 1.2$ range to the communication probability of the nodes.

The cardinality algorithm inherently is not affected by the noise. However, the empirical failure detection use cases we provided slightly deteriorate in the face of noisy data. In our experiments, we saw that for the AS-733 and Airports graphs, the accuracy decreased by 10% on average and up to 35% for some feature-label combinations in AS-733 graph. Overall, the algorithms are still able to successfully detect changes in most cases. Sophisticated detection approaches can be implemented to improve quality and make it even more robust to noise, however, this is not in the scope of this paper.

## REFERENCES


[1] A. E. Sarıyüce, E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Regularizing graph centrality computations," *JPDC*, vol. 76(C), pp. 106–119, 2015.

[2] G. S. Malkin, "Rip version 2," Internet Requests for Comments, RFC Editor, STD 56, Nov. 1998, http://www.rfc-editor.org/rfc/rfc2453.txt.

[3] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comp. & System Sci.*, 31(2), pp. 182–209, 1985.

[4] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. 16th Intl. Conf. Extending Database Technology*, 2013.

[5] Ö. Şimşek and A. G. Barto, "Skill characterization based on betweenness," in *Advances in neural information processing systems*, 2009, pp. 1497–1504.

[6] S. Jin, Z. Huang, Y. Chen, D. G. Chavarría-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *IPDPS*, 2010, pp. 1–7.

[7] E. L. Merrer and G. Trédan, "Centralities: Capturing the fuzzy notion of importance in social graphs," in *Proc. @nd ACM EuroSys W. Social Network Systems (SNS)*, 2009.

[8] Z. Shi and B. Zhang, "Fast network centrality analysis using GPUs," *BMC Bioinformatics*, vol. 12, p. 149, 2011.

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

[10] M. Riondato and E. Upfal, "Abra: Approximating betweenness centrality in static and dynamic graphs with rademacher averages," in KDD '16. New York, NY, USA: ACM, 2016, pp. 1145–1154.

[11] Y. Jia, V. Lu, J. Hoberock, M. Garland, and J. C. Hart, "Edge v. node parallelism for graph centrality metrics," *GPU Computing Gems*, vol. 2, pp. 15–30, 2011.

[12] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. G. Chavarría-Miranda, "A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets," in *IPDPS W. Multithreaded Architectures and Applications*, 2009.

[13] E. Cohen, "All-distances sketches, revisited: Hip estimators for massive graphs analysis," *IEEE TKDE,*, vol. 27, no. 9, pp. 2320–2334, 2015.

[14] P. Boldi and S. Vigna, "In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond," *CoRR*, vol. abs/1308.2144, 2013.

[15] B. W. Priest and G. Cybenko, "Approximating centrality in evolving graphs: toward sublinearity," pp. 10 184 – 10 184 – 9, 2017.

[16] M. Y. Özkaya, A. E. Sarıyüce, Ü. V. Çatalyürek, and A. Pınar, "Active betweenness cardinality: Algorithms and applications," ArXiv, Tech. Rep. arXiv:1711.10634, Nov 2017.

[17] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Networks*, vol. 3, no. 32, pp. 245–251, 2010.

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