Efficient Monte Carlo Algorithms for Approximating Katz Centrality on Large Graphs

Garrett W. Cornett
Texas State University
San Marcos, TX, USA
gwc38@txstate.edu

Minh Phu Vuong
Texas State University
San Marcos, TX, USA
cty13@txstate.edu

Chul-Ho Lee*
Texas State University
San Marcos, TX, USA
chulho.lee@txstate.edu

Abstract

Katz centrality is an important metric for measuring the relative importance of nodes in a graph, commonly used in social networks. This measure is prohibitively expensive to compute exactly, as it requires up to cubic-time complexity. Although several approximation methods exist, most fail to exploit parallelism, limiting their scalability to large graphs. To address this limitation, we propose two highly parallelizable Monte Carlo algorithms based on random walks named MC-Katz and MC-KatzP, which approximate the truncated Katz centrality and its original form, respectively. Empirical results on nine real-world datasets show that they achieve low mean relative error and high accuracy in ranking the most influential nodes in the networks, while delivering two to three orders of magnitude speedups over the baselines.

CCS Concepts

• **Mathematics of computing → Probabilistic algorithms**; **Graph algorithms**.

Keywords

Katz centrality, Monte Carlo algorithms, GPU parallel computing

ACM Reference Format:
Garrett W. Cornett, Minh Phu Vuong, and Chul-Ho Lee. 2026. Efficient Monte Carlo Algorithms for Approximating Katz Centrality on Large Graphs. In *Proceedings of the Nineteenth ACM International Conference on Web Search and Data Mining (WSDM '26), February 22–26, 2026, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3773966.3779378

1 Introduction

As social networks become more commonplace and complex, the need to analyze and make inferences about such networks has increased rapidly. While many underlying structural characteristics exist for researchers to study, one of the most ubiquitous is that of centrality measures, metrics that can be used to measure the relative importance and significance of nodes in a graph. Although there are a variety of different centrality metrics based on how one defines “importance” [2, 3, 5, 7, 16], in this paper we focus on Katz centrality, a variation of eigenvector centrality.

*This work was supported by NSF under Grants 2209921 and 2209922, the International Energy Joint R&D Program of the Korea Institute of Energy Technology Evaluation and Planning (No. 20228530050030), and an equipment donation from NVIDIA.

Motivation. Katz centrality [12] measures the influence of a node in a graph by calculating the weighted sum of the walks from all nodes to that node in the graph, with longer walks penalized by an attenuation factor. It has been used across various domains, such as learning protein-protein interactions [8], link prediction [22, 24], vulnerability scanning and detection [23], and strategic marketing [20]. However, this is prohibitively expensive to compute exactly, as it requires matrix inversion that generally has a complexity of $O(n^3)$. More efficient approximation algorithms exist, but few lend themselves to parallelism, limiting their scalability.

Related Work. Foster *et al.* [6] proposed a power-method-based approximation, which is currently used in the NetworkX implementation [10]. Nathan *et al.* [17] devised an agglomerative approach to approximating matrix powers for Katz centrality by utilizing a variant of breadth-first search to count the number of walks between nodes. Recently, Lin *et al.* [14] introduced SAKE, a novel uniform sampling-based estimator to approximate Katz centrality. However, these approaches focus on improving performance on the CPU and do not scale well with parallelization. In contrast, we develop efficient Monte Carlo (MC) algorithms based on random walks to approximate Katz centrality, which can be readily parallelized on the GPU, thereby achieving scalable performance.

Contributions. Our contributions are as follows: (1) We propose a parallel random walk-based MC framework to estimate the Katz centrality of a graph. (2) From the framework, we develop two estimators: MC-Katz, a fixed-length truncated walk estimator, and MC-KatzP, a full walk estimator derived from a novel probabilistic interpretation of Katz centrality. (3) We empirically confirm the accuracy of our estimators using mean relative error (MRE) and similarity scores of high-influence nodes over nine real-world datasets. (4) We compare our serial estimators to SAKE, achieving speedups of two to three orders of magnitude. (5) We demonstrate the advantage of the parallelism inherent in our estimators by comparing their parallel implementation against the NetworkX-cuGraph implementation, achieving up to 184× speedup.

2 Katz Centrality

Consider a strongly connected directed graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of nodes and E is the set of edges. The graph G is characterized by an $n \times n$ adjacency matrix $\mathbf{A} = [A_{ij}]$, where $A_{ij} = 1$ if there is an edge from node j to node i, and $A_{ij} = 0$ otherwise. We write $\mathbf{A}^k$ to denote the k-th power of $\mathbf{A}$ and $[\mathbf{A}^k]_{ij}$ to denote the (i, j)-entry of $\mathbf{A}^k$, which indicates the number of walks of length k from j to i. Finally, let $d(i)$ be the out-degree of node i.

Katz centrality computes the influence of nodes in a graph by counting the number of walks of any length that end at each node, multiplied by an attenuation factor at each length of the walk.

Specifically, the Katz centrality of node i is defined by

$$K_i = \beta + \beta \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k [\mathbf{A}^k]_{ij}, \tag{1}$$

where k is the length of the walk ending at node i, $\beta > 0$ is introduced to ensure that every node has a non-zero centrality, and $\alpha \in (0, 1)$ is the attenuation factor. Note that the value of α needs to be chosen such that $\alpha < 1/\lambda(\boldsymbol{A})$ to guarantee convergence, where $\lambda(\boldsymbol{A})$ is the spectral radius of the graph. Also, since α^k exponentially decreases with k, it is common practice to truncate the Katz centrality by limiting walk length k to a finite length k', i.e.,

$$K_i \approx \tilde{K}_i := \beta + \beta \sum_{k=1}^{k'} \sum_{j=1}^{n} \alpha^k [\mathbf{A}^k]_{ij}, \tag{2}$$

Katz centrality can also be calculated by solving a system of linear equations, as is similarly done for the PageRank algorithm [9]. Specifically, letting $\boldsymbol{K} = [K_1, K_2, \ldots, K_n]$ to denote a vector of the Katz centrality values of nodes, we can write

$$\boldsymbol{K} = \alpha \mathbf{A} \boldsymbol{K} + \beta \boldsymbol{e}, \tag{3}$$

which has a closed-form solution $\boldsymbol{K} = \beta(\mathbf{I} - \alpha\mathbf{A})^{-1}\boldsymbol{e}$ if $\alpha < 1/\lambda(\boldsymbol{A})$, where $\mathbf{I}$ is the $n \times n$ identity matrix, and $\boldsymbol{e}$ is the all-one vector [18].

3 Proposed Method

3.1 Mathematical Framework

Random Walk Estimator. Below, we demonstrate that $[\mathbf{A}^k]_{ij}$ can be estimated by r independent realizations of a k-step Markov chain, or a k-step random walk, that moves on G. Observe that $[\mathbf{A}^k]_{ij}$, the number of k-length walks from j to i in the graph G, is the same as the number of k-length walks from i to j in its *transposed* graph. Also, the Katz centrality of each node is based on its in-neighbors' centrality values, as in (1), whereas a random walk moves from the current node to one of its out-neighbors. Thus, hereafter we consider the *transposed* version of G for the sake of convenience. In other words, $A_{ij} = 1$ indicates the presence of an edge from i to j, and $[\mathbf{A}^k]_{ij}$ indicates the number of walks of length k from i to j.

We define a Markov chain $\{X_t\}_{t \geq 0}$ on the graph G, where X_t denotes the state (node) of the Markov chain at time t. Its transition matrix is given by $\mathbf{P} = [P_{ij}]$, where P_{ij} represents the transition probability from node i to node j, i.e., $P_{ij} = \mathbb{P}\{X_{t+1} = j \mid X_t = i\}$. We focus on the simple random walk as a Markov chain on G, where, at each step, it moves from the current node to one of its out-neighbors that is chosen uniformly at random. That is, the transition probability from i to j is given by $P_{ij} = A_{ij}/d(i)$.

Consider r independent realizations of a k-step simple random walk that originates from node i, each of which is now used as a random *sample* of a k-length walk originating from i to estimate $[\mathbf{A}^k]_{ij}$. However, we cannot simply leverage a fraction of sampled k-length walks that start from i and end at j to estimate $[\mathbf{A}^k]_{ij}$. This is because the probability of having (or sampling) a specific sequence of visited nodes, say, $(i, i_1, \ldots, i_{k-1}, i_k)$, is not uniform but $1/(d(i)d(i_1)\cdots d(i_{k-1}))$. In other words, k-length walks passing through low-degree nodes would be sampled more often than those having high-degree nodes. Thus, a sampling bias is introduced when naively using the r independent realizations to estimate $[\mathbf{A}^k]_{ij}$.

To correct the bias, we reweight each sampled k-length walk by the inverse of its sampling probability. Specifically, letting $X_t^{(\ell)}$, $t = 0, 1, \ldots, k$, be the node at time t of the ℓ-th realization of k-step simple random walk, where $\ell = 1, 2, \ldots, r$, we define

$$Z_{ij}^{(\ell)} = \frac{\mathbf{1}\{X_k^{(\ell)} = j\}}{1/\prod_{t=0}^{k-1} d(X_t^{(\ell)})} = \mathbf{1}\{X_k^{(\ell)} = j\} \prod_{t=0}^{k-1} d(X_t^{(\ell)}), \tag{4}$$

where $\mathbf{1}\{X_k^{(\ell)} = j\}$ is the indicator function that returns 1 if the ℓ-th realization visits node j after the k-th step. Then, we can build the following MC estimator $\widehat{[\mathbf{A}^k]}_{ij}$ for $[\mathbf{A}^k]_{ij}$:

$$\widehat{[\mathbf{A}^k]}_{ij} = \frac{1}{r}\sum_{\ell=1}^{r} Z_{ij}^{(\ell)} = \frac{1}{r}\sum_{\ell=1}^{r} \mathbf{1}\{X_k^{(\ell)} = j\} \prod_{t=0}^{k-1} d(X_t^{(\ell)}). \tag{5}$$

After computing $\mathbb{E}[Z_{ij}^{(\ell)}]$ from (4) by conditioning on the ℓ-th realization of k-step simple random walk, and by the strong law of large numbers [4], we can show that

$$\widehat{[\mathbf{A}^k]}_{ij} \xrightarrow{\text{a.s.}} [\mathbf{A}^k]_{ij}, \text{ as } r \to \infty. \tag{6}$$

That is, the estimator $\widehat{[\mathbf{A}^k]}_{ij}$ is unbiased. It is also worth noting that this estimator can be regarded as an importance sampling estimator, since it involves reweighting by the inverse probability.

MC-Katz Estimator. We first introduce MC-Katz, a simple yet effective MC estimator to approximate the truncated Katz centrality in (2), which is built upon $\widehat{[\mathbf{A}^k]}_{ij}$ in (5). Specifically, by substituting $\widehat{[\mathbf{A}^k]}_{ij}$ for $[\mathbf{A}^k]_{ij}$ in (2), we have the following MC-Katz estimator:

$$\hat{K}_i = \beta + \beta \sum_{k=1}^{k'} \sum_{j=1}^{n} \alpha^k \widehat{[\mathbf{A}^k]}_{ij}. \tag{7}$$

Here, each summand $\widehat{[\mathbf{A}^k]}_{ij}$ can be obtained by using the r independent realizations of k'-step simple random walk up to k steps, as in (5). Since each summand $\widehat{[\mathbf{A}^k]}_{ij}$ converges to $[\mathbf{A}^k]_{ij}$ almost surely, for $k = 1, 2, \ldots, k'$, as r goes to infinity, $\hat{K}_i$ also converges to $\tilde{K}_i$ almost surely, i.e., $\hat{K}_i$ is unbiased. When it comes to the implementation of MC-Katz, r sample paths of k'-step simple random walk just need to be generated independently, thereby making the implementation highly parallelizable and allowing us to leverage GPU parallel processing effectively.

MC-KatzP Estimator. We next introduce MC-KatzP, an MC estimator to approximate the (original) Katz centrality in (1). To this end, we observe that the Katz centrality of node i can be interpreted probabilistically, with the expected numbers of L-length walks from j to i, where L is geometrically distributed with parameter $1 - \alpha$. Formally, we have the following result:

LEMMA 3.1. *The Katz centrality of node i in* (1) *can be written as*

$$K_i = \beta + \alpha\beta \sum_{j=1}^{n} \mathbb{E}\left[\sum_{k=1}^{L} [\mathbf{A}^k]_{ij}\right], \tag{8}$$

where the expectation is with respect to a geometric random variable L with parameter $1-\alpha$, i.e., $\mathbb{P}\{L = k\} = \alpha^{k-1}(1-\alpha)$, $k = 1, 2, \ldots$.

PROOF. Recall that the Katz centrality of node i is given by

$$K_i = \beta + \beta \sum_{j=1}^{n} \sum_{k=1}^{\infty} \alpha^k [\mathbf{A}^k]_{ij} = \beta + \alpha\beta \sum_{j=1}^{n} \sum_{k=1}^{\infty} \alpha^{k-1} [\mathbf{A}^k]_{ij}. \tag{9}$$

Algorithm 1: Parallel Implementation of MC-Katz

```
Input  : graph $G(V, E)$, attenuation factor $\alpha$, constant term
         $\beta$, number of walks per node $r$, walk length $k'$
Output : estimated Katz centrality vector $\hat{K}$
1  for each thread tid $\in \{0, 1, \dots, rn - 1\}$ in parallel do
2      $v_c \leftarrow \mathrm{tid}/r$
3      $s_{\mathrm{tid}} \leftarrow 0$
4      $d_{\mathrm{prod}} \leftarrow 1$
5      for $k = 1, 2, \dots, k'$ do
6          $d_{\mathrm{prod}} \leftarrow d(v_c) \times d_{\mathrm{prod}}$
7          $v_c \leftarrow \mathrm{RandomSelect}(N(v_c))$
8          $s_{\mathrm{tid}} \leftarrow (\alpha^k \times d_{\mathrm{prod}}) + s_{\mathrm{tid}}$
9      $\mathrm{atomicAdd}(\hat{K}[\mathrm{tid}/r], s_{\mathrm{tid}})$
10 for each thread tid $\in \{0, 1, \dots, n - 1\}$ in parallel do
11     $\hat{K}[\mathrm{tid}] \leftarrow \beta \times (1 + \hat{K}[\mathrm{tid}]/r)$
```

By noting that $\alpha^{k-1} = \mathbb{P}\{L \geq k\}$ for $k = 1, 2, \dots$, we have

$$\sum_{k=1}^{\infty} \alpha^{k-1} [\mathbf{A}^k]_{ij} = \sum_{k=1}^{\infty} \mathbb{P}\{L \geq k\} [\mathbf{A}^k]_{ij} = \sum_{k=1}^{\infty} \sum_{\ell=k}^{\infty} \mathbb{P}\{L = \ell\} [\mathbf{A}^k]_{ij}$$
$$= \sum_{\ell=1}^{\infty} \mathbb{P}\{L = \ell\} \sum_{k=1}^{\ell} [\mathbf{A}^k]_{ij} = \mathbb{E}\left[\sum_{k=1}^{L} [\mathbf{A}^k]_{ij}\right], \quad (10)$$

where the expectation is with respect to L. Substituting (10) into (9) yields (8). □

Similar to $[\widehat{\mathbf{A}^k}]_{ij}$ in (5), we can build an MC estimator to approximate $\mathbb{E}\big[\sum_{k=1}^{L} [\mathbf{A}^k]_{ij}\big]$ based on r independent sample paths of L-step simple random walk, where the walk length of each sample path is now drawn from the geometric distribution with parameter $1 - \alpha$. By using Lemma 3.1 and substituting this estimator for $\mathbb{E}\big[\sum_{k=1}^{L} [\mathbf{A}^k]_{ij}\big]$ in (8), we have the MC-KatzP estimator to approximate the (original) Katz centrality in (1). We omit the details due to space constraints.

3.2 Parallel Implementation

We implement MC-Katz to leverage the inherent parallelism of (7), which we exploit using GPU hardware and the CUDA programming framework [19]. Algorithm 1 outlines the pseudocode of MC-Katz. A k'-step random walk is assigned to a single thread using thread-level parallelism, with each serving as an independent realization of $Z_{ij}^{(\ell)}$, originating from node v_c, to contribute to $[\widehat{\mathbf{A}^k}]_{ij}$ in (5) and thus to $\hat{K}_i$ in (7). This level of parallelism helps ensure full GPU utilization, as the number of threads launched is rn, which is often a sufficient workload to keep all GPU cores occupied even at smaller graph sizes. MC-KatzP is similarly implemented; however, instead of using a set length k', each thread generates its walk length from the geometric distribution with parameter $1-\alpha$ in order to estimate $\mathbb{E}\big[\sum_{k=1}^{L} [\mathbf{A}^k]_{ij}\big]$, as in (8).

We also introduce several optimizations in our implementation. Threads are assigned a starting node on a blocked scheduling scheme to maximize cache utilization, and we avoid thread divergence by using a set walk length. That is, as seen in Line 2 of Algorithm 1, r contiguous threads start their random walks from the same node in the graph. We empirically observe that this blocked scheduling scheme is more efficient than a cyclic scheduling scheme, where r contiguous threads start their random walks from *different* nodes, given the irregular data access patterns inherent to random walks. Furthermore, we offload the β addition and multiplication operations to the walk normalization step, which significantly reduces the number of operations. Similarity, we replace the bias correction operation that involves the inverse probability weighting with the one with multiplications, as outlined in (4), which execute considerably faster. Additionally, we utilize the Xoshiro256++ random number generator [1] in our implementation, as it outperformed the built-in CUDA random number generators for our purposes.

Table 1: Graph statistics

Dataset	# Nodes	# Edges	Avg. Degree
soc-Epinions1 (SE)	75,879	508,837	6.71
slashdot0902 (SD)	82,168	948,464	11.54
email-EuAll (EE)	265,214	420,045	1.58
web-NotreDame (WN)	325,729	1,497,134	4.60
web-Stanford (WS)	281,903	2,312,497	8.20
web-Google (WG)	875,713	5,105,039	7.86
wiki-Talk (WT)	2,394,385	5,021,410	2.10
Pokec (PK)	1,632,803	30,622,564	6.32
LiveJournal (LJ)	4,847,571	68,993,773	14.23

4 Performance Evaluation

4.1 Setup

Datasets. We consider nine real-world unweighted, directed network datasets from the SNAP repository [13], listed in Table 1 by ascending node count. For each dataset, we use the transposed graph in the implementations of MC-Katz and MC-KatzP, as explained in Section 3.1. Additionally, we use the largest strongly connected component of each graph where self-loops are removed.

Software and hardware. MC-Katz and MC-KatzP are implemented in C++ and CUDA[1], and compiled using the nvcc compiler (version 12.4). All experiments are conducted on a Linux server equipped with an Intel Xeon Gold 5218R CPU, 96 GB RAM, and an NVIDIA RTX Quadro 8000 48-GB GPU.

Baselines. For performance comparison, we consider two baselines: SAKE [14] and NetworkX-cuGraph [21]. SAKE is the state-of-the-art sampling-based estimator for Katz centrality, which uses random node sampling and counts walks on the sampled subgraph. To match its serial design, we use the serial versions of MC-Katz and MC-KatzP for comparison. When it comes to comparing our parallel estimators to NetworkX-cuGraph, we enable its cuGraph backend so that the computation of Katz centrality runs on parallel C++/CUDA kernels, and we report only the runtime of the accelerated function.

Parameter setting. By default, we set the parameters r, the number of random walks per node, and k', the length of simple random walk for MC-Katz, to 1000 and 6, respectively, and use them unless otherwise specified. We also set the value of α to $1/n$, ensuring $\alpha < 1/\lambda(A)$, since $\lambda(A) \leq d_{\max}$ for any connected graph G, where $d_{\max}$ is the maximum degree [11]. The value of β is set to 1 in all cases. For SAKE, we set the sampling rate and the number of loops to 0.01 and 20, respectively.

Performance metrics. To empirically demonstrate the efficiency of MC-Katz and MC-KatzP, we measure the execution times of our serial and parallel implementations, SAKE and NetworkX-cuGraph, which we use to calculate the speedups of MC-Katz and MC-KatzP.

[1]https://github.com/Garrett-Cornett/MonteCarlo-Katz

Table 2: Mean relative errors of MC-Katz and MC-KatzP

	SE	SD	EE	WN	WS	WG	WT	PK	LJ
MC-Katz	2.87×10^{-8}	1.15×10^{-8}	8.75×10^{-9}	1.34×10^{-9}	1.50×10^{-10}	7.27×10^{-12}	8.63×10^{-9}	2.78×10^{-11}	1.15×10^{-11}
MC-KatzP	1.99×10^{-6}	9.21×10^{-7}	6.23×10^{-7}	2.18×10^{-7}	2.60×10^{-7}	1.74×10^{-7}	3.93×10^{-7}	8.11×10^{-10}	1.28×10^{-10}
SAKE	1.29×10^{-5}	3.45×10^{-6}	5.16×10^{-6}	1.87×10^{-6}	8.92×10^{-7}	1.74×10^{-7}	2.65×10^{-6}	8.11×10^{-10}	–

Table 3: Runtimes of serial implementations (in seconds)

Dataset	SAKE	MC-KatzP (S)	MC-Katz (S)
SE	533.090	1.618	4.687
SD	2813.643	3.584	12.270
EE	233.097	1.720	4.672
WN	167.152	2.706	6.868
WS	1280.159	7.579	20.990
WG	9097.057	21.838	61.897
WT	26025.269	5.631	20.879
PK	212609.417	65.695	456.908
LJ	>3 days	192.801	1243.444

Table 4: Runtimes of parallel implementations (in seconds)

Dataset	NX-cuGraph	MC-KatzP	MC-Katz
SE	2.0536	0.0210	0.0174
SD	3.1360	0.0417	0.0399
EE	1.3010	0.0218	0.0184
WN	1.6832	0.0325	0.0278
WS	5.4881	0.0836	0.0781
WG	12.2684	0.2017	0.2150
WT	5.5287	0.0630	0.0643
PK	105.9564	0.6192	1.2334
LJ	273.9694	1.4871	2.5372

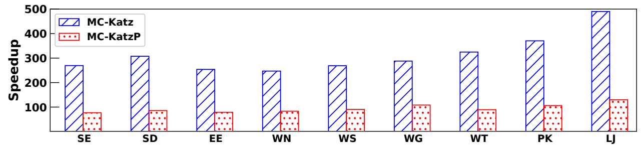
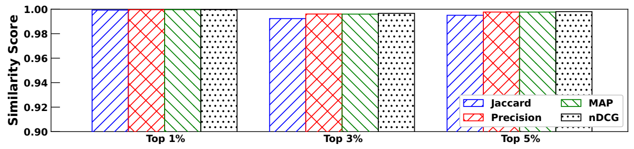


Figure 1: Speedups of parallel over serial implementations.

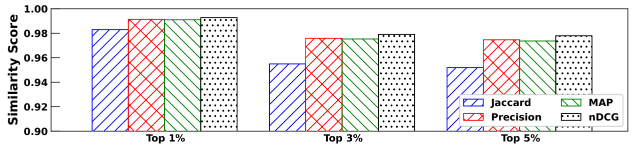
For accuracy, we calculate the MRE of the results of MC-Katz and MC-KatzP. We also calculate similarity scores of the top ranked nodes in the graph based on Katz centrality values, using the Jaccard index, precision, mean average precision (MAP), and normalized discounted cumulative gain (nDCG) [15]. We choose to compare the top 1% ranked nodes by default, as this offers a more robust view of similarity across different graph sizes, compared to a constant value such as top-100 nodes [14]. To compute MRE and similarity scores, the results of NetworkX-cuGraph are used as the ground truth, and all Katz centrality values are normalized using the l^2-norm.

4.2 Performance Comparison

Comparison with SAKE. We first demonstrate the efficiency and accuracy of MC-Katz and MC-KatzP by comparing their serial implementations against SAKE. As shown in Table 3, MC-Katz and MC-KatzP are two to three orders of magnitude faster when approximating the Katz centrality values of the entire graph. For the largest dataset (LJ), serial MC-KatzP is able to complete its approximation in just over 3 minutes, while SAKE is unable to complete its approximation in less than 72 hours. This demonstrates SAKE's lack of scalability, as the overhead needed to construct the sampled subgraph grows with increasing graph size. As seen in Table 2, both MC-Katz and MC-KatzP are able to achieve lower MRE rates than SAKE, with an average MRE of 6.10×10^{-10} and 9.18×10^{-8} for MC-Katz and MC-KatzP, respectively, compared



(a) MC-Katz



(b) MC-KatzP

Figure 2: Similarity scores of MC-Katz and MC-KatzP.

to 7.84×10^{-7} for SAKE. Our implementations are also designed with parallelization in mind, with parallel MC-Katz and parallel MC-KatzP achieving average speedups of 313.3× and 94.4×, respectively, when compared to their serial versions, as seen in Figure 1.

Comparison with NetworkX-cuGraph. We next demonstrate the efficiency and accuracy of parallel MC-Katz and MC-KatzP compared to the NetworkX-cuGraph counterpart. As shown in Table 4, both MC-Katz and MC-KatzP demonstrate significant speedups compared to the counterpart, yielding average speedups of 81.68× and 94.90× across the datasets, respectively. For accuracy, we observe average similarity scores of 99.96% and 98.95% for MC-Katz and MC-KatzP, respectively, when ranking the top 1% of nodes. In addition, both implementations provide approximations with low MRE across all datasets, as seen in Table 2. Notably, as the size of the graph increases, the MREs of the estimators decrease, indicating that the accuracy of our estimators increases on larger graphs.

MC-Katz vs. MC-KatzP. When considering the parallel implementations, MC-Katz shows higher efficiency than MC-KatzP on smaller datasets, as the latter involves an additional operation of generating a geometric random variate on each thread and utilizes different random walk lengths. However, on larger datasets, MC-KatzP begins to outperform MC-Katz, as the reduced workload of the shorter walks begins to offset the overhead of generating the random variate. Figure 2 shows the average similarity scores when ranking the top 1%, 3%, and 5% of nodes compared to the ground truth. Both estimators can predict the rankings of the most important nodes with a high degree of accuracy, with MC-Katz achieving higher accuracy than MC-KatzP. We attribute this to the truncated walks of MC-Katz capturing the contributions of distant nodes better than the randomly generated walk lengths of MC-KatzP, where longer walks are exceedingly rare events.

5 Conclusion

In this paper, we have introduced MC-Katz and MC-KatzP, which provide highly parallelizable random walk-based estimations of Katz centrality, and demonstrated their superior performance with experimental evaluation. Both MC-Katz and MC-KatzP exhibit marked speedup when compared to SAKE and NetworkX-cuGraph, while maintaining a high accuracy in their estimations.

References

[1] David Blackman and Sebastiano Vigna. 2021. Scrambled linear pseudorandom number generators. *ACM Transactions on Mathematical Software (TOMS)* 47, 4 (2021), 1–32.

[2] Francis Bloch, Matthew O Jackson, and Pietro Tebaldi. 2023. Centrality measures in networks. *Social Choice and Welfare* 61, 2 (2023), 413–453.

[3] Kousik Das, Sovan Samanta, and Madhumangal Pal. 2018. Study on centrality measures in social networks: A survey. *Social Network Analysis and Mining* 8, 1 (2018), 1–11.

[4] Rick Durrett. 2019. *Probability: Theory and examples.* Cambridge University Press, USA.

[5] David Easley and Jon Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge University Press, USA.

[6] Kurt Foster, Stephen Muth, John Potterat, and Richard Rothenberg. 2001. A Faster Katz Status Score Algorithm. *Computational & Mathematical Organization Theory* 7 (12 2001), 275–285.

[7] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau. 2023. Graph Vulnerability and Robustness: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5915–5934.

[8] Ziqi Gao, Chenran Jiang, Jiawen Zhang, Xiaosen Jiang, Lanqing Li, Peilin Zhao, Huanming Yang, Yong Huang, and Jia Li. 2023. Hierarchical graph learning for protein–protein interaction. *Nature Communications* 14, 1 (2023), 1093.

[9] David Gleich, Leonid Zhukov, and Pavel Berkhin. 2004. Fast parallel PageRank: A linear system approach. *Yahoo! Research Technical Report* 13 (2004), 22.

[10] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference* (Pasadena, California). 11–15.

[11] Roger A Horn and Charles R Johnson. 2012. *Matrix analysis.* Cambridge University Press, USA.

[12] Leo Katz. 1953. A New Status Index Derived from Sociometric Analysis. *Psychometrika* 18, 1 (1953), 39–43.

[13] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[14] Mingkai Lin, Wenzhong Li, Lynda J Song, Cam-Tu Nguyen, Xiaoliang Wang, and Sanglu Lu. 2021. Sake: Estimating katz centrality based on sampling for large-scale social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 4 (2021), 1–21.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *Introduction to information retrieval.* Cambridge University Press, USA.

[16] C. Mavroforakis, M. Mathioudakis, and A. Gionis. 2015. Absorbing random-walk centrality: Theory and algorithms. In *Proceedings of IEEE International Conference on Data Mining* (Atlantic City, NJ, USA). IEEE, 901–906.

[17] Eisha Nathan and David A Bader. 2018. Approximating personalized Katz centrality in dynamic graphs. In *Parallel Processing and Applied Mathematics: 12th International Conference.* Springer, Springer International Publishing, Cham, 290–302.

[18] Mark Newman. 2018. *Networks.* Oxford University Press, USA.

[19] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable parallel programming with CUDA. In *ACM SIGGRAPH 2008 Classes.* Association for Computing Machinery, New York, NY, USA, 1–14.

[20] Long Ren, Bin Zhu, and Zeshui Xu. 2021. Robust consumer preference analysis with a social network. *Information Sciences* 566 (2021), 379–400.

[21] RAPIDS Development Team. 2023. *RAPIDS: Libraries for End to End GPU Data Science.* https://rapids.ai

[22] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. 2022. Graph neural networks: Foundation, frontiers and applications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* Association for Computing Machinery, New York, NY, USA, 4840–4841.

[23] Yueming Wu, Deqing Zou, Shihan Dou, Wei Yang, Duo Xu, and Hai Jin. 2022. Vulcnn: An image-inspired scalable vulnerability detection system. In *Proceedings of the 44th International Conference on Software Engineering.* Association for Computing Machinery, New York, NY, USA, 2365–2376.

[24] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems* 34 (2021), 29476–29490.