Efficient and Accurate PageRank Approximation on Large Graphs

SIYUE WU, Shenzhen University, China
DINGMING WU*, Shenzhen University, China
JUNYI QUAN, Shenzhen University, China
TSZ NAM CHAN, Shenzhen University, China
KEZHONG LU, Shenzhen University, China

PageRank is a commonly used measurement in a wide range of applications, including search engines, recommendation systems, and social networks. However, this measurement suffers from huge computational overhead, which cannot be scaled to large graphs. Although many approximate algorithms have been proposed for computing PageRank values, these algorithms are either (i) not efficient or (ii) not accurate. Worse still, some of them cannot provide estimated PageRank values for all the vertices. In this paper, we first propose the CUR-Trans algorithm, which can reduce the time complexity for computing PageRank values and has lower error bound than existing matrix approximation-based PageRank algorithms. Then, we develop the T^2-Approx algorithm to further reduce the time complexity for computing this measurement. Experiment results on three large-scale graphs show that both the CUR-Trans algorithm and the T^2-Approx algorithm achieve the lowest response time for computing PageRank values with the best accuracy (for the CUR-Trans algorithm) or the competitive accuracy (for the T^2-Approx algorithm). Besides, the two proposed algorithms are able to provide estimated PageRank values for all the vertices.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**; **Approximation algorithms analysis**.

Additional Key Words and Phrases: Graph Algorithms, Approximation Algorithms, PageRank

ACM Reference Format:
Siyue Wu, Dingming Wu, Junyi Quan, Tsz Nam Chan, and Kezhong Lu. 2024. Efficient and Accurate PageRank Approximation on Large Graphs. *Proc. ACM Manag. Data* 2, 4 (SIGMOD), Article 196 (September 2024), 26 pages. https://doi.org/10.1145/3677132

1 Introduction

PageRank was originally proposed to evaluate the importance of the web pages in search engines [12]. It has wide applications in web search engines, social networks [10], recommendation systems [68], and so on. In general, PageRank is designed to measure the importance of vertices within the context of the entire graph.

A desired approximate PageRank algorithm should efficiently compute and accurately estimate PageRank values on large graphs. However, existing approximate PageRank algorithms are either

Authors' Contact Information: Siyue Wu, wusiyue1229@gmail.com, Shenzhen University, China; Dingming Wu, dingming@szu.edu.cn, Shenzhen University, China; Junyi Quan, 2210274031@email.szu.edu.cn, Shenzhen University, China; Tsz Nam Chan, edisonchan@szu.edu.cn, Shenzhen University, China; Kezhong Lu, kzlu@szu.edu.cn, Shenzhen University, China.

*Corresponding author

ACM 2836-6573/2024/9-ART196
https://doi.org/10.1145/3677132

inefficient or inaccurate. Table 1 compares representative approximate PageRank algorithms. Specifically, iteration algorithms take $O\left(m \cdot (\log n + \log_\alpha \tau)\right)$ [20, 63] time on a graph with m edges and n vertices. Since they terminate and return the results when the difference between the PageRank vectors from the latest two consecutive iterations is below a given threshold τ, the error bound is τ. However, they take $O(\log n + \log_\alpha \tau)$ iterations to satisfy the termination condition that is time-consuming on large graphs. Monte Carlo-based algorithms have to perform $N = O\left(\frac{n \log n}{\epsilon^2}\right)$ α-random walks to achieve a good relative accuracy with high probability, which is expensive. Sampling-based algorithms estimate PageRank values based on local graph structures. The time complexity is determined by the size of the used local graph structure. However, they cannot provide approximate PageRank values for the vertices outside the local graph structures. Existing matrix approximation-based algorithms compute low-rank approximation of the transition matrix of the graph. Although the computation costs on the original transition matrix have been reduced to the computation costs on the low-rank matrix, the time complexity of constructing low-rank matrices is high. Moreover, the error bounds of existing matrix approximation-based algorithms are large.

Table 1. Comparisons of different approximate PageRank algorithms.

Method	Algorithm	Error bound	Time complexity
Iteration	Power iteration algorithm [63]	$\|\pi - \tilde{\pi}\|_1 = \tau$	$O(k \cdot m)$ [20], $k = O(\log n + \log_\alpha \tau)$
	PMSI [77]		unknown
	NL1 [3]		$O(d^2 \cdot n \cdot \log^2 n + (d^2 \log(n/d^2 + 1))/\tau^2)$
Monte Carlo	Monte Carlo End-point with Random Start [11]	$P[\|\pi - \tilde{\pi}\|_1 \le \epsilon] \ge n - \frac{\sum_j var(\tilde{\pi}_j)^2}{\epsilon^2}$, $var(\tilde{\pi}_j) = O(\frac{\pi_j(1-\pi_j)}{N})$, $N = O(\frac{n \cdot \log n}{\epsilon^2})$	$O(\frac{n \cdot \log n}{\alpha \cdot \epsilon^2})$
	Monte Carlo Complete Path/ End-point with Cyclic Start [4]		
Sampling	ApproxRank [76]	$\|\pi - \tilde{\pi}\|_1 \le \frac{\alpha}{1-\alpha} \left\|EXT - \widetilde{EXT}\right\|_1$	$O(n + k \cdot m_s)$, $k = O(\log n_s + \log_\alpha \tau)$
	Local PageRank algorithms [1, 14]	unknown	$O(n + k \cdot m_s)$, $k = O(\log n_s + \log_\alpha \tau)$
Matrix approximation	Direct sampling in power iteration (DSPI) [51]	$\|T^k - \tilde{T}^k\|_2 \le k\eta\phi\|T\|_F^k, \eta > 0$	$O(m + k \cdot m_s)$, $k = O(\log n_s + \log_\alpha \tau)$
	Adaptive sampling in power iteration (ASPI) [51]	$\|T^k - \tilde{T}^k\|_2 \le \eta \frac{a(1-a^k)}{1-a} \|T\|_F^k, \eta > 0$	$O(k \cdot m + \sum_k m_s^{(k)})$, $k = O(\log n_s + \log_\alpha \tau)$
	SVD-based PageRank approximation [7]	$\|T - \tilde{T}\|_F^2 \le \|T - T_\rho\|_F^2 + \varepsilon\|T\|_F^2$ [24]	$O(\rho \cdot n^2 + (c^2 + \rho^2) \cdot n + c^3 + k \cdot \rho^2)$, $k = O(\log \rho + \log_\alpha \tau)$
	CUR-Trans (ours)	$\|T - CUR\|_F \le (1 + \phi)\|T - T_\rho\|_F$, where $\phi = \varphi^2 + 2\varphi$, $\varphi = \sqrt{\rho \cdot c} + \sqrt{c}$	$O((c \cdot r + c^2) \cdot n + r^2 c + r^3 + c^2 r + c^3 + k \cdot c^2)$, $k = O(\log c + \log_\alpha \tau)$
	T^2**-Approx (ours)**	$\|T^2 - X \cdot Y\|_F \le \sqrt{(n-c)}\|T\|_F^2$	$O(c^2 \cdot n + k \cdot c^2)$, $k = O(\log c + \log_\alpha \tau)$

Notes: the descriptions of the symbols are available in Table 2.

To tackle the issues, we propose a transformation model that generalizes existing matrix approximation-based PageRank algorithms. It builds the connection between the low-rank approximation and the random walk, so that any low-rank approximation algorithm can be incorporated to estimate PageRank values. Using the transformation model, we propose the CUR-Trans algorithm that estimates PageRank values based on a low-rank approximation of the transition matrix based on the CUR decomposition [26]. To the best of our knowledge, this is the first work to consider the CUR decomposition in the PageRank approximation. Comparing with existing matrix approximation-based PageRank algorithms, our CUR-Trans algorithm has lower time complexity (Lemma 1) and better error bound (Lemma 3).

Although the low-rank approximate matrix is good at preserving the information in the original matrix, the time complexity of constructing a low-rank approximate matrix is bounded by the time complexity of the matrix decomposition algorithm, e.g., the CUR decomposition takes $O(c^2 \cdot r + c^3)$ time [49]. Based on the observation that approximating matrix multiplication [23] takes $O(c)$ time

that is much more efficient than decomposing a matrix, we propose the T^2-Approx algorithm that approximates the multiplication of two transition matrices, i.e., $T^2 \approx X \cdot Y$, and estimates PageRank values based on the low-rank approximation of T^2. The error bound of T^2-Approx (Lemma 4) is similar to that of existing algorithms, whereas the time complexity of T^2-Approx (Lemma 5) is lower than that of existing algorithms and that of the CUR-Trans algorithm.

Finally, extensive experiments are conducted on three large real-world graph datasets to evaluate the performance of our algorithms. The results show that our algorithms CUR-Trans and T^2-Approx are able to provide good estimations on the PageRank values of all the vertices when using small sampling ratios. They outperform the competitors in terms of the computation time. The error of CUR-Trans is better than that of T^2-Approx, while T^2-Approx is more efficient than CUR-Trans.

As a remark, there are also many other studies for utilizing parallel/distributed computation [42, 54, 55, 82] to further accelerate PageRank computation. In this work, we focus on developing approximate algorithms on single machine setting with CPU. Despite that, the computation time of our algorithms only takes 100 seconds on graphs with hundreds of millions of vertices and billions of edges. We leave the combination of our algorithms with parallel/distributed optimization opportunities in our future work.

To summarize, the main contributions of the paper are as follows:

- We propose a general model, called the transformation model, for estimating PageRank values based on the low-rank approximation of the transition matrix. It can incorporate any low-rank approximation algorithm for PageRank approximation.
- We first consider the CUR decomposition in the PageRank approximation and propose the CUR-Trans algorithm with better error bound.
- We first adopt matrix multiplication approximation to estimate PageRank values and propose the T^2-Approx algorithm with better time complexity.
- Extensive evaluations on large real-world graph datasets demonstrate that our algorithms significantly outperform the state-of-the-art algorithms.

The rest of the paper is organized as follows : Section 2 reviews existing approximate PageRank algorithms. Relevant concepts and techniques are provided in Section 3. Section 4 proposes the transformation model and the CUR-Trans algorithm. The T^2-Approx algorithm is presented in Section 5. Extensive evaluations are conducted in Section 6 and we conclude in Section 7.

2 Related Work

Approximate PageRank algorithms can be classified into the following five categories.

Iteration methods. Some studies aim to reduce the number of vertices involved in the computation. The Quadratic Extrapolation algorithm [41] accelerates the convergence of the iteration process by periodically subtracting estimates of the non-principal eigenvectors from the current iteration. Based on the observation that many vertices converge to their true PageRank values quickly, while only a few vertices are slow-converging, the Adaptive PageRank algorithm [39] accelerates the PageRank computation by removing the converged vertices at each iteration. However, it cannot guarantee that all vertices can correctly converge.

Some studies convert the PageRank problem into a linear system of equations and investigate different ways of solving the linear system. Based on the observation that the smaller the damping factor is, the easier it is to solve the PageRank problem, the inner/outer stationary algorithm [28] designs inner and outer iterations to solve a linear system with similar algebraic structure to the original one, but with a lower damping factor. Parameters are carefully chosen to trade off the numbers of inner and outer iterations and the approximation accuracy. The power-inner–outer (PIO) iteration algorithm [31] combines the inner/outer stationary algorithm and the power iteration

algorithm. RTSS [78] and PMSI [77] further improve the PIO algorithm by introducing more parameters to reduce the number of iterations. The GIO algorithm [69] is a general version of the inner/outer stationary algorithm. Corso et al. [19] rearrange the coefficient matrix of the linear system to increase the data locality and reduce the number of iterations. For bounded-degree graphs, the NL1 algorithm [3] converts the PageRank problem into a convex minimization problem over a unit simplex, and then solve it using iterative methods.

Some studies compute approximate PageRank values based on partitioned graphs. The BlockRank algorithm [40] first computes the local PageRank values in each partition and estimates the importance of each partition. Then, it calculates approximate PageRanks by aggregating the local PageRank values and the importance of partitions. Algorithms B_LIN and NB_LIN [70] generate k graph partitions by using graph partitioning algorithms METIS [43]. Let matrix $W_{1,i}$ contain the edges within partition i and W_2 contain the edges across partitions. The approximate PageRank values are calculated based on the inverses of the matrices $\{W_{1,i}\}$ of k partitions and a low-rank approximation of W_2.

Although the above iteration methods can improve the efficiency for computing PageRank values, these methods can only be applied on an entire graph, which suffer from high computation cost for large-scale graphs. Compared with these methods, our methods perform iterations on a low-rank approximate matrix, which can be more scalable to support very large graphs.

Monte Carlo methods. This type of method calculates approximate PageRank values by simulating random walks on the graph. The end-point with random start algorithm [11] simulates s runs of the random walk initiated at a randomly chosen vertex. The PageRank value of a vertex is estimated by the fraction of the random walks that end at it. The estimation accuracy is determined by the number of random walks. To reduce the variance of the estimation, the end-point with cyclic start algorithm [4] simulates $s \cdot n$ runs of the random walk initiated at each vertex s times. For same purpose, the complete path algorithms [4] simulate the random walk s times from each vertex and estimates the PageRank value of vertex v based on the total number of visits to v. The Push algorithm [17] can be regarded as a de-randomized version of the Monte Carlo method. Each vertex has a residual value r and a PageRank value p. In each iteration, each vertex passes an amount of r to its neighbours and stores the rest amount of r in its own p. Finally, for each vertex, the value of p is taken as the PageRank value.

Even though various Monte Carlo methods have been proposed in the literature, all these methods have to undergo a large number of iterations in order to achieve a relatively good accuracy with high probability. As such, these methods cannot be scalable to large graphs.

Sampling-based methods. The sampling-based algorithms estimate PageRank values based on sampled subgraphs. The local PageRank algorithm [14] expands a small subgraph around the target vertex and uses this subgraph as the basis for the estimation of the PageRank value. Bar-Yossef and Mashiach [1] improve the local PageRank algorithm by a pruning technique that removes the vertices with small influence in each iteration. ApproxRank [76] computes approximate PageRank values on a subgraph. It represents the set of vertices outside the subgraph with an external vertex Λ and extends the subgraph with links to Λ. Then, random walks are conducted on the modified transition matrix with respect to Λ.

Although sampling-based methods are efficient for approximately computing PageRank values of sampling vertices, these methods cannot find the PageRank values of those vertices that are not inside the sample set. Compared with these methods, our method can estimate the PageRank values of all vertices.

Matrix approximation-based methods. This type of methods estimates PageRank values based on low-rank approximation of the transition matrix. Benczúr et al. [7] construct a low-rank approximate matrix based on the linear time SVD decomposition [24]. Liu et al. [51] construct low-rank

approximate matrices by conducting element-wise matrix sampling. They propose two algorithms. The Direct Sampling in Power Iteration (DSPI) constructs one approximate matrix. The Adpative Sampling in Power Iteration (ASPI) adaptively adjusts the sampling rate to construct a new approximate matrix in each iteration.

The above studies use specific ways to produce low-rank approximate transition matrices. We propose a general model that can incorporate any low-rank approximation algorithm and a new efficient algorithm for generating a low-rank approximation of the transition matrix.

Personalized PageRank methods. Personalized PageRank (PPR) [15, 50, 59, 72] is a variant of PageRank, which computes the importance of vertices with respect to a specific vertex or seed set of vertices. The PPR values of the vertices can be used to derive the global PageRank values [73]. Specifically, given a graph with n vertices, let $\pi(s,t)$ be the PPR value of vertex t w.r.t. vertex s, which denotes the probability that an α-random walk starting from vertex s terminates at vertex t. The global PageRank value of vertex t is calculated as $\pi_t = \sum_{s=1}^{n} \pi(s,t)/n$ where the PPR value of vertex t w.r.t. any vertex $\pi(\cdot,t)$ can be obtained by issuing a single target PPR query. The time complexity of the best known single target PPR algorithm is $O\left(\frac{\pi(t)\cdot n \log n}{\alpha\epsilon}\right)$ [50]. Then, it takes $O\left(\frac{\pi(t)\cdot n^2 \log n}{\alpha\epsilon}\right)$ time to calculate the global PageRank values of all the vertices by calling the PPR algorithm, which is not a practical solution on large graphs.

3 Preliminaries

This section presents important concepts of PageRank and relevant techniques used in the proposed algorithms. Table 2 lists the symbols used in the paper.

PageRank. Given a graph $G = (V, E)$, $|V| = n$, $|E| = m$, let T be its transition matrix and outdeg(i) be the out-degree of vertex v_i. In T, if $(v_i, v_j) \in E$, $T_{i,j} = 1/\text{outdeg}(i)$, otherwise, $T_{i,j} = 0$. According to the surfer model of web browsing [47], in a graph consisting of pages and links, a web surfer randomly opens a page. He jumps to any linked page according to the transition matrix T with α probability and jumps to any unlinked page with $1-\alpha$ probability, where α is called damping factor and is often set to 0.85. Eventually, page browsing will have a stationary distribution π, which contains the PageRank values of all the pages. It indicates the likelihood of each page being visited. Equation 1 shows the formulation of PageRank [63].

$$\pi = \left(\alpha \cdot T + (1-\alpha) \cdot [\frac{1}{n}]_{n\times n}\right) \cdot \pi. \tag{1}$$

Based on simple mathematical derivations, the PageRank vector π is shown as follows [8]:

$$\pi = (1-\alpha) \cdot \left(\sum_{k=0}^{\infty} \alpha^k T^k\right) \cdot \pi_0, \tag{2}$$

where $\pi_0 = [\frac{1}{n}]_{n\times 1}$ is the initial PageRank vector.

The PageRank vector π is usually computed using the power iteration method [8, 34, 63]. Given a termination threshold τ, the power iteration method computes

$$\pi^{(k+1)} = \alpha \cdot T \cdot \pi^{(k)} + (1-\alpha) \cdot \pi_0, \tag{3}$$

until $\|\pi^{(k+1)} - \pi^{(k)}\|_2 < \tau$.

Euclidean norm based sampling probability. Given a matrix T, the Euclidean norm [35] of columns j in T is calculated as $\sqrt{\sum_{i=1}^{n} |T_{i,j}|^2}$ and the Euclidean norm of row i in T is calculated as $\sqrt{\sum_{j=1}^{n} |T_{i,j}|^2}$. In a matrix, there are three kinds of sampling probability distributions based on the Euclidean norms of columns and rows [23]:

Table 2. Symbols.

Symbols	Description
T, $T_{n\times n}$	Transition matrix.
$\tilde{T}$	A low-rank approximation of T.
G, G^T	Graph, transformed graph.
V	Vertex set.
E	Edge set.
n, m	The number of vertices, the number of edges.
ρ	Matrix rank.
c	The number of sampled columns. Also the number of vertices in the transformed graph.
r	The number of sampled rows.
π, $\tilde{\pi}$	PageRank vector, approximate PageRank vector.
π_0	Initial PageRank vector.
α	The damping factor.
$\|x\|_2$	The Euclidean norm of a vector x, $\|x\|_2 = \sqrt{\sum x_i^2}$.
$\|T\|_F$	The Forbenius norm of a matrix T, $\|T\|_F = \sqrt{\sum_i \sum_j x_{i,j}^2}$.
$[\frac{1}{n}]_n$	A column vector of size $n \times 1$.
k	The number of iterations.
m_s	The number of edges in a sampled graph.
n_s	The number of vertices in a sampled graph.
$m_s^{(k)}$	The number of edges in a sampled graph at the k^{th} iteration.
d	The maximum number of non-zero elements in a row or column.
EXT ($\widetilde{EXT}$)	The vector containing the accurate (approximate) PageRank values of the external vertices of subgraph.
ϵ	An error.
τ	The termination threshold in the iteration method.

- The Euclidean norm square of column probability distribution $\mathcal{F}(p^c)$: the probability of selecting column j.

$$p_j^c = \frac{\sum_{i=1}^n |T_{i,j}|^2}{\sum_{k=1}^n \sum_{i=1}^n \left|T_{i,k}\right|^2} \tag{4}$$

- The Euclidean norm square of row probability distribution $\mathcal{F}(p^r)$: the probability of selecting row i.

$$p_i^r = \frac{\sum_{j=1}^n |T_{i,j}|^2}{\sum_{k=1}^n \sum_{j=1}^n \left|T_{k,j}\right|^2} \tag{5}$$

- The Euclidean norm of column and row probability distribution $\mathcal{F}(p^{cr})$: the probability of simultaneously selecting column i and row i.

$$p_i^{cr} = \frac{\sqrt{\sum_{j=1}^n T_{j,i}^2}\sqrt{\sum_{j=1}^n T_{i,j}^2}}{\sum_{k=1}^n \left[\sqrt{\sum_{j=1}^n T_{j,k}^2}\sqrt{\sum_{j=1}^n T_{k,j}^2}\right]} \tag{6}$$

Low-rank approximation. Given a matrix T and a rank ρ, low-rank approximation aims to find a matrix $\tilde{T}_\rho$ whose rank is ρ, such that $\|T - \tilde{T}_\rho\|$ is minimized. Matrix $\tilde{T}_\rho$ is a rank-ρ approximation of T [27]. Matrix decomposition algorithms such as SVD [49] and CUR [26] can be used to construct low-rank approximate matrices. If a matrix T can be decomposed using the SVD, i.e., $T = U\Sigma V^T$, a rank-ρ approximation of T can be constructed based on the first ρ singular vectors and singular values, i.e., $\tilde{T}_\rho = U_{n\times\rho}\Sigma_{\rho\times\rho}V^T_{\rho\times n}$ [13, 29]. The CUR decomposition can also be used to construct a rank-ρ approximate matrix, i.e., $\tilde{T}_\rho = CU_\rho R$, where the rank of U_ρ is ρ.

4 Transformation-based Method

We introduce the transformation model in Section 4.1. Sections 4.2 and 4.3 present the CUR-Trans algorithm and its error bound, respectively.

4.1 Transformation Model

According to the power iteration method for PageRank computation (Equation 2), performing k iteration steps can be considered as k-step random walks on graph G. To efficiently calculate the PageRank values of the vertices in G, we propose the transformation model that converts the random walks on a large graph G to the random walks on a small transformed graph G^T.

Specifically, the transformation model first decomposes $T_{n\times n}$ into two matrices, such that $T_{n\times n} \approx A_{n\times c} \cdot B_{c\times n}$, $c \ll n$. Then, it constructs a transformed graph G^T with c vertices by considering the transition matrix $D_{c\times c}$ with small size (as c is small), where $D_{c\times c} = B_{c\times n} \cdot A_{n\times c}$. Next, the initial PageRank vector π_0 of the original graph is transformed into a vector π_T using matrix B. Vector π_T is taken as the initial PageRank vector of the transformed graph G^T. Then, a traditional PageRank algorithm is applied on the transformed graph G^T and we obtain the PageRank values of the c vertices in G^T. Lastly, the PageRank vector of the transformed graph is converted back to the PageRank vector of the original graph G using matrix A. Figure 1 shows the working processing of the transformation model.

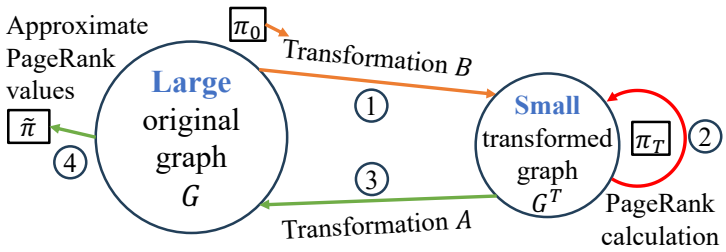


Fig. 1. Transformation model.

Correctness. Below, we prove the correctness of the transformation model by converting the original equation of PageRank (Equation 2) based on T into the formula based on D.

Given $T_{n\times n} \approx A_{n\times c} \cdot B_{c\times n}$ and $D_{c\times c} = B_{c\times n} \cdot A_{n\times c}$, we have

$$\begin{aligned}
\pi &= (1-\alpha)\cdot\left(\sum_{k=0}^{\infty}\alpha^k T^k\right)\cdot\pi_0 \approx (1-\alpha)\cdot\left(\sum_{k=0}^{\infty}\alpha^k (A\cdot B)^k\right)\cdot\pi_0 \\
&= (1-\alpha)\cdot\pi_0 + (1-\alpha)\cdot\left(\sum_{k=1}^{\infty}\alpha^k A\cdot(B\cdot A)^{k-1}\cdot B\right)\cdot\pi_0 \\
&= (1-\alpha)\cdot\pi_0 + A\cdot\alpha(1-\alpha)\cdot\left(\sum_{k=1}^{\infty}\alpha^{k-1} D^{k-1}\right)\cdot B\cdot\pi_0 \\
&= (1-\alpha)\cdot\pi_0 + \alpha\cdot A\cdot\underline{(1-\alpha)\cdot\left(\sum_{k'=0}^{\infty}\alpha^{k'} D^{k'}\right)\pi_T}, \qquad (7)
\end{aligned}$$

where $k' = k-1$ and $\pi_T = B\cdot\pi_0$ that is a vector of size $c\times 1$. Equation 2 means that taking π_0 as the initial PageRank vector, k-step random walks are performed based on transition matrix T. The underlined part in Equation 7 means that taking π_T as the initial PageRank vector, k'-step random walks are performed based on transition matrix D. The result of the underlined part in Equation 7 is the PageRank values in the transformed graph G^T and is transformed back to the PageRank vector of size $n\times 1$ by multiplied by matrix A.

The transformation model generalizes existing matrix approximation based PageRank algorithms [7, 51]. One can adopt different matrix decomposition algorithms to construct the matrices A and B in the transformation model. However, the exact SVD algorithm takes $O(n^3)$ time for $n\times n$ matrices [49]. Well-known scalable SVD algorithms are approximate algorithms, listed in Table 3. Nevertheless, most of these matrix decomposition approximate algorithms are either inefficient or only applicable to specific types of matrices. According to Table 3, CUR is more efficient than the other matrix decomposition algorithms. Thus, we propose the CUR-based transformation that finds an approximate solution of $T \approx A\cdot B$ in the next section. Then, the outputs of the CUR-based transformation are approximate PageRank values.

Table 3. Time complexity of different matrix decomposition algorithms.

Algorithm	Time complexity	Requirements
QR [52]	$O(n^3)$ [80]	Invertible matrix
LU/PLU [64]	$O(n^3/3)$ [53]	Invertible matrix
EVD [81]	$O(n^3)$ [45]	Square matrix
Nyström [46]	$O((c\cdot\rho)\cdot n + c^3)$ [48]	Symmetric and positive definite matrix
Full Rank [65]	$O(n^2)$ [6]	Non-invertible matrix
SVD [30]	$O(n^3)$ [49]	None
Truncated SVD [44]	$O(r\cdot n^2)$	None
LinearTime SVD [24]	$O(c^2\cdot n + c^3)$	None
ConstantTime SVD [24]	$O(c^3 + c\cdot r^2)$	None
Randomized SVD [60]	$O(n^2 log(n) + r^3)$	None
CUR [22]	$O(c^2\cdot r + c^3)$ [26, 49]	None

4.2 CUR-based Transformation

We propose the CUR-Trans algorithm that is based on the CUR decomposition to construct the transformed graph mentioned in the previous section and calculates approximate PageRank values. Algorithm 4.1 shows the pseudo code of algorithm CUR-Trans. It first constructs matrix $C_{n\times c}$ by sampling c columns from T with replacement from the Euclidean-norm of column probability distribution, i.e., Equation 4 (line 1). Then, it constructs matrix $R_{r\times n}$ by sampling r rows from T with replacement from the Euclidean-norm of row probability distribution, i.e., Equation 5 (line 2). Line 3 calculates matrix W by taking the intersection of matrices R and C. Next, it computes a rank-x approximation of U based on W, denoted as $\tilde{U}$ (line 4). Deriving matrix $\tilde{U}$ is a standard step in the CUR decomposition. Specifically, if W can be decomposed, matrix U is the Moore–Penrose pseudo inverse of W and $\tilde{U}$ is the low-rank approximation matrix of U by taking the first x singular values of U. So far, the transition matrix T is decomposed into three matrices based on the CUR decomposition, i.e., $T_{n\times n} \approx C_{n\times c} \cdot \tilde{U}_{c\times r} \cdot R_{r\times n}$. After that, we set $A \leftarrow C$, $B \leftarrow \tilde{U} \cdot R$, and $D \leftarrow B \cdot A$ (line 5). Then, the PageRank vector π_0 of size $n \times 1$ is initialized as $\left[\frac{1}{n}\right]_n$ and is transformed into a vector $\tilde{\pi}_T$ of size $c \times 1$ using matrix B (line 6). Next, the traditional PageRank algorithm such as the power iteration algorithm is applied and the PageRank vector $\tilde{\pi}_T^{(k)}$ of the transformed graph D is obtained (lines 7–10). In the end, vector $\tilde{\pi}_T^{(k)}$ is transformed back to a vector of size $n \times 1$ using matrix A (line 11). After normalization, vector $\tilde{\pi}$ contains the approximate PageRank values in the original graph G and is returned (lines 12 and 13). Note that, it is equivalent to set $A \leftarrow C \cdot \tilde{U}$, $B \leftarrow R$ at line 5 and all results of CUR-Trans hold.

Algorithm 4.1 CUR-Trans

Input: The transition matrix T of graph G containing n vertices, the number of sampled columns c, the number of sampled rows r, termination threshold τ, the damping factor α.
Output: Approximate PageRank values $\tilde{\pi}$ of G.

1: Construct C by sampling c columns from T with replacement based on the probability $\{p_j^c\}_{j=1}^n$ in Equation 4;
2: Construct R by sampling r rows from T with replacement based on the probability $\{p_i^r\}_{i=1}^n$ in Equation 5;
3: $W \leftarrow R \cap C$;
4: $\tilde{U} \leftarrow$ get a rank-x approximation of U based on W;
5: $A \leftarrow C, B \leftarrow \tilde{U} \cdot R, D \leftarrow B \cdot A$;
6: $\pi_0 \leftarrow \left[\frac{1}{n}\right]_n, \tilde{\pi}_T^{(0)} \leftarrow B \cdot \pi_0, k \leftarrow 0$;
7: **repeat**
8: $\quad k \leftarrow k + 1$;
9: $\quad \tilde{\pi}_T^{(k)} \leftarrow \alpha \cdot D \cdot \tilde{\pi}_T^{(k-1)} + (1-\alpha) \cdot \left[\frac{1}{c}\right]_c$;
10: **until** $\|\tilde{\pi}_T^{(k)} - \tilde{\pi}_T^{(k-1)}\|_2 < \tau$
11: $\tilde{\pi} = (1-\alpha) \cdot \pi_0 + \alpha \cdot A \cdot \tilde{\pi}_T^{(k)}$;
12: $\tilde{\pi} \leftarrow$ normalizing $\tilde{\pi}$;
13: **return** $\tilde{\pi}$;

Example 1. Figure 2 demonstrates the process of algorithm CUR-Trans using an example graph. The original graph G contains 10 vertices. Matrix C is constructed by the 4 sampled columns 1, 5, 6, 8 (yellow columns) in T. Matrix R is constructed by the 4 sampled rows 6, 7, 8, 9 (green rows) in T. Then, matrices W and $\tilde{U}$ are calculated. According to algorithm CUR-Trans, matrix C

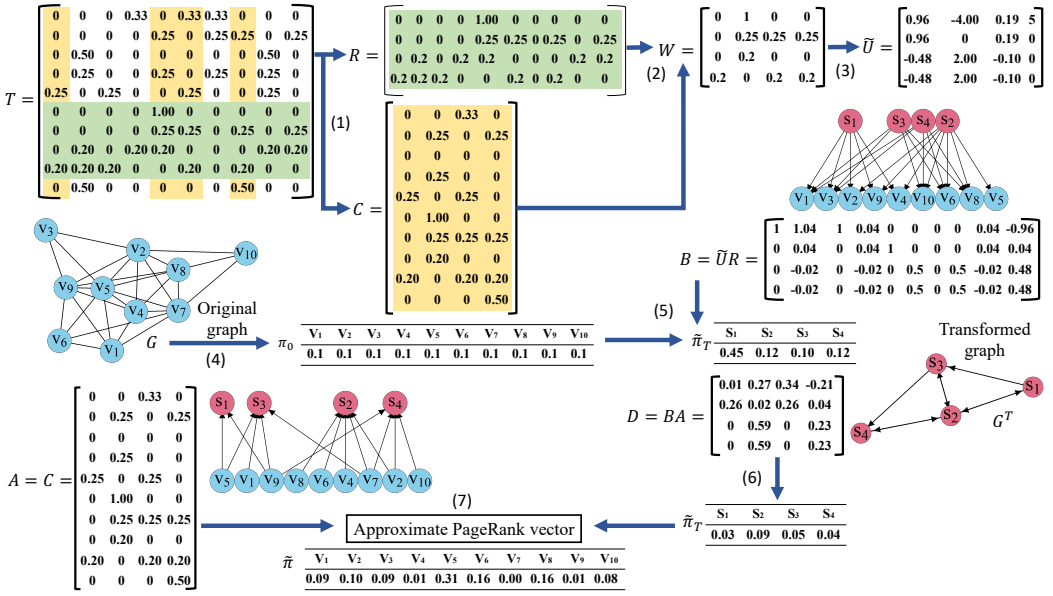


Fig. 2. CUR-based transformation.

is taken as the transformation matrix A and transformation matrix B is calculated as $\tilde{U} \cdot R$. Then, the transformed graph G^T with transition matrix $D = B \cdot A$ is constructed, as shown in the figure. Next, the PageRank values of the 10 vertices in G are initialized as 0.1. The PageRank vector π_0 of G is converted into the PageRank vector $\tilde{\pi}_T$ of the transformed graph G^T using matrix B. Then, the power iteration algorithm is applied and the PageRank values of the 4 vertices in the transformed graph are obtained. Finally, the approximate PageRank values of the 10 vertices in the original graph G are derived by transforming the PageRank vector $\tilde{\pi}_T$ back using matrix A.

Discussion. In the CUR-Trans algorithm, the transition matrix T is decomposed using modified CUR decomposition that differs from the traditional CUR decomposition as follows:

(i) Traditional CUR decomposition conducts sampling using the distribution based on T-singular vectors that achieves better theoretical error bound [26, 56]. However, the traditional CUR decomposition is inefficient because the T-singular vectors are derived based on the SVD decomposition. Our algorithm based on the Euclidean norm of column and row probability distributions (Equations 4 and 5) has a slightly larger error bound (Lemma 3), but improves the efficiency by avoiding this expensive SVD decomposition.

(ii) For each sampled column and row, the traditional CUR decomposition performs a scaling operation that divides each element in the column and the row by $\sqrt{c \cdot p_i^c}$ and $\sqrt{r \cdot p_i^r}$, respectively. The purpose of applying the scaling operation in the traditional CUR is to get a better theoretical error bound for the matrix decomposition [22, 25, 26]. We are the first to adopt the CUR decomposition for PageRank estimation and we found that applying scaling operation cannot produce better estimated results and is time-consuming. The transformed graph preserves the important information embedded in the original graph. Applying the scaling operation on the transformed graph is equivalent to changing the edge weights, so that it may result in inaccurate estimation. Thus, our algorithm does not use the scaling operation.

(iii) In the traditional CUR decomposition, matrix U is the Moore-Penrose pseudo-inverse of W. Specifically, matrix W is firstly decomposed by the SVD decomposition, i.e., $W = H \cdot \Sigma \cdot V^T$. Then, matrix U is calculated as $U = V \cdot \Sigma^{-1} \cdot H^T$. However, the SVD decomposition is time-consuming when matrix W is large. To reduce the computational cost, the modified CUR decomposition approximates U by taking the first x singular values and singular vectors of W (line 4 in the CUR-Trans algorithm), i.e., $U \approx \tilde{U} = V_x \cdot \Sigma_x^{-1} \cdot H_x^T$.

Below, we explain the effectiveness of the rank-x approximation of U. Given a desired rank ρ, by setting $c = r = O(\rho \cdot \log(\rho))$ according to Lemma 3, matrix CUR is a low-rank approximation of T. The rank of CUR is usually much higher than the desired rank ρ. Figure 3 shows the desired rank ρ and the rank of CUR on dataset Orkut that is used in the experiment. Thus, it is unnecessary to spend computational cost on the SVD decomposition of W. Given the desired rank ρ, according to $c = r = O(\rho \cdot \log(\rho))$, we set $x = \sqrt{c}$. Then, matrix $\tilde{U}$ is a low-rank approximation of U. Thus, the rank of matrix $C\tilde{U}R$ is $\log c$. As shown in Figure 3, on dataset Orkut, the rank of $C\tilde{U}R$ matches the desired rank ρ perfectly. Hence, the modified CUR decomposition efficiently computes a rank-ρ approximation of T.

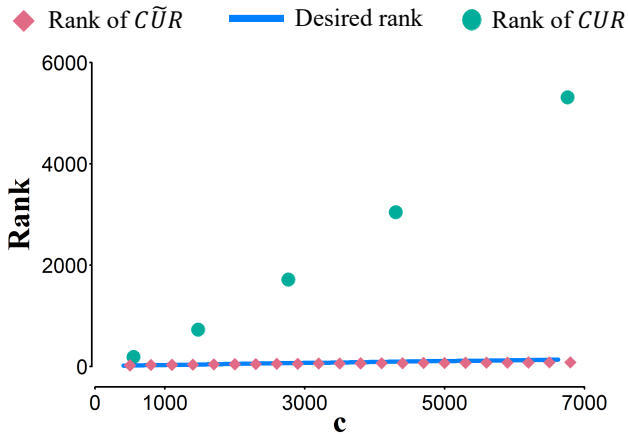


Fig. 3. Effectiveness of rank-x approximation of U.

Lemma 1. *The time complexity of the CUR-Trans algorithm is* $O((c^2 + c \cdot r) \cdot n + c^2 \cdot (\log c + \log_\alpha \tau) + c^2 \cdot r + c^3)$, *where* $c, r \ll n$.

Proof. The computational cost of the CUR-Trans algorithm is dominated by three phases. In the first phase, constructing matrices C, R and W takes $O(r + c)$ time (lines 1–3). According to [49], constructing matrix U takes $O(c^2 \cdot r + c^3 + c \cdot (c \cdot r + r))$ time. Then, constructing matrix $\tilde{U}$ (line 4) takes $O(c^2 \cdot r + c^3 + x \cdot (c \cdot r + r)) = O(c^2 \cdot r + c^3)$, since $x = \sqrt{c}$. In the second phase (lines 5 and 6), constructing matrices A, B, and D takes $O(c^2 \cdot n + c \cdot r \cdot n)$ time. Since the number k of iterations needed on matrix $T_{n \times n}$ is $O(\log n + \log_\alpha \tau)$ [63], the number of iterations needed on the transformed graph is $O(\log c + \log_\alpha \tau)$. Given that the time complexity of the power iteration method on a graph having m edges is $O(k \cdot m)$ [61, 62], we derive that the third phase (lines 10–13) takes $O(c^2 \cdot (\log c + \log_\alpha \tau))$ time to compute the PageRank values in the transformed graph, since the number of edges in the transformed graph is bounded by c^2. Overall, the time complexity of the CUR-Trans algorithm is $O(c^2 \cdot n + c \cdot r \cdot n + c^2 \cdot (\log c + \log_\alpha \tau) + c^2 \cdot r + c^3)$, where $c, r \ll n$. □

4.3 Error Bound

How to analyze error bound for matrix approximation-based PageRank algorithms? Consider previous algorithms [7, 51] that compute approximate PageRank values $\tilde{\pi}$ using approximate transition matrices $\tilde{T}$. In their analysis, the error of the approximate PageRank vector $\|\pi - \tilde{\pi}\|_2$ is bounded by the error of the k^{th} power of the approximate matrix $\|T^k - \tilde{T}^k\|_F$, i.e.,

$$\|\pi - \tilde{\pi}\|_2 \leq (1-\alpha) \cdot \left(\sum_{k=1}^{\infty} \alpha^k \|T^k - \tilde{T}^k\|_F \right) \cdot \left\| [\frac{1}{n}]_n \right\|_F. \tag{8}$$

However, the bounds derived based on $\|T^k - \tilde{T}^k\|_F$ are large, as shown in Table 1. Based on the observation that matrix T^k has the property that the sum of each row is 1, we present Lemma 2 that bounds $\|T^k - \tilde{T}^k\|_F$ using $\|T - \tilde{T}\|_F$ as follows.

Lemma 2. *Given two transition matrices T and $\tilde{T}$ of the same size $n \times n$,*

$$\|T^k - \tilde{T}^k\|_F \leq k \cdot \sqrt{n} \cdot \|T - \tilde{T}\|_F.$$

The proof is available in the supplementary material.

According to Equation 8 and Lemma 2, we have

$$\begin{aligned} \|\pi - \tilde{\pi}\|_2 &\leq (1-\alpha) \cdot \left(\sum_{k=1}^{\infty} k \cdot \alpha^k \right) \cdot \sqrt{n} \cdot \|T - \tilde{T}\|_F \cdot \left\| [\frac{1}{n}]_n \right\|_F \\ &= \frac{\alpha}{1-\alpha} \cdot \sqrt{n} \cdot \|T - \tilde{T}\|_F \cdot \left\| [\frac{1}{n}]_n \right\|_F. \end{aligned} \tag{9}$$

Then, given a matrix approximation-based PageRank algorithm, one just needs to derive the bound of $\|T - \tilde{T}\|_F$, so that the error of the estimated PageRank vector is bounded. According to Equation 9, the error bound is independent of the number of iterations k.

Error bound of CUR-Trans. Since we use the modified CUR decomposition in the CUR-Trans algorithm, the error bound of the traditional CUR decomposition can not be applied. Below, we provide the error bound of the CUR-Trans algorithm in Lemma 3.

Lemma 3. *Given a graph G, let T be its transition matrix. Given $\rho (\leq rank(T))$, let T_ρ be the optimal ρ-rank approximation of T. If algorithm CUR-Trans sets the number of sampled columns and rows as $c = r = O(\rho \cdot \log(\rho))$, we have*

$$\Pr\left(\|T - CUR\|_F \leq (1+\phi) \|T - T_\rho\|_F \right) \geq 0.81, \tag{10}$$

where $\phi = \varphi^2 + 2\varphi$, $\varphi = \sqrt{\rho \cdot c} + \sqrt{c}$.

The proof is available in the supplementary material.

Discussion. The error bound [26] of the traditional CUR decomposition is: given $0 < \epsilon < 1$ and $c = r = O((\rho \log(\rho))/\epsilon^2)$,

$$\Pr\left(\|T - CUR\|_F \leq (1+\epsilon) \|T - T_\rho\|_F \right) \geq 0.49. \tag{11}$$

Although the error bound of the CUR-Trans algorithm (Equation 10) is larger than that of the traditional CUR decomposition (Equation 11), the probability of the bound of the CUR-Trans algorithm is higher.

We proceed to compare the error bound of the CUR-Trans algorithm with that of existing matrix approximation-based algorithms listed in Table 1. According to Lemma 2 and simple mathematical derivations, we have $\|T^k - (CUR)^k\|_2 < \|T^k - (CUR)^k\|_F < k\sqrt{n}\|T - CUR\|_F$. Based on Equation 10, we have $\|T^k - (CUR)^k\|_2 < k\sqrt{n}(1+\phi)\|T - T_\rho\|_F$. Considering the error bounds of DSPI and ASPI

in Table 1, their bounds are based on $\|T\|_F^k, \|T\|_F \leq 1$ that is larger than our bound based on $\|T - T_\rho\|_F$. Considering the SVD-based PageRank approximation algorithm in Table 1, because $\varepsilon > 1$, $\left(\|T - T_\rho\|_F^2 + \varepsilon\|T\|_F^2\right)^{1/2} > (\|T - T_\rho\|_F + \|T\|_F)/\sqrt{2}$ and $\|T - T_\rho\|_F < \|T\|_F$, we have $(\|T - T_\rho\|_F + \|T\|_F)/\sqrt{2} < (1 + \phi)\|T - T_\rho\|_F$. Therefore, its error bound is larger than our bound.

5 T^2-Approximation Method

This section explains the weakness of existing matrix approximation-based method in Section 5.1. Sections 5.2 and 5.3 present the T^2-Approx algorithm and its error bound, respectively.

5.1 Motivation

According to the equation for computing the PageRank vector (Equation 2), calculating T^k dominates the time cost of the PageRank computation. The time complexity of calculating T^k is $O(k \cdot n^3)$ [18, 33, 75]. Existing matrix approximation-based method [7] reduces the time complexity to $O(k \cdot c^3)$ by converting the multiplication of matrix T of size $n \times n$ to the multiplication of a small matrix of size $c \times c$, where $c \ll n$. Specifically, it approximates T by a low-rank matrix $\tilde{T}$ via decomposing T into two matrices X and Y, such that $T_{n\times n} \approx \tilde{T}_{n\times n} = X_{n\times c} \cdot Y_{c\times n}$. Then, T^k can be approximated as

$$\begin{aligned} T^k \approx \tilde{T}^k = (XY)^k &= \underbrace{(XY) \cdot (XY) \cdot (XY) \cdots (XY)}_{k} \\ &= X \cdot \underbrace{(YX) \cdot (YX) \cdots (YX)}_{k-1} \cdot Y = X \cdot Z^{k-1} \cdot Y, \end{aligned} \tag{12}$$

where $Z = Y \cdot X$ is a small matrix of size $c \times c$.

One may adopt different matrix decomposition algorithms listed in Table 3 to implement Equation 12. However, these matrix decomposition algorithms are either inefficient or only applicable to specific types of matrices. Based on the observation that approximating matrix T is inefficient, we propose the T^2-Approx algorithm that efficiently approximates matrix T^2 in the next section.

5.2 T^2-Approx Algorithm

According to the matrix multiplication conversion (MMC) algorithm [21–23], it is efficient to approximate the multiplication of two matrices by the multiplication of anther two matrices, i.e., $A \cdot B \approx C \cdot D$. It can be applied to any type of matrix and its time complexity is $O(c)$ [23] where c is the number of sampled columns and rows. Taking the advantage of the MMC algorithm, we propose to approximate the multiplication of two T by the multiplication of another two matrices, i.e.,

$$T^2_{n\times n} = T_{n\times n} \cdot T_{n\times n} \approx X_{n\times c} \cdot Y_{c\times n}, \tag{13}$$

where $c \ll n$. Then, we consider T^k as the multiplication of several T^2, and thus T^k can be approximated as

$$\begin{aligned} T^k = \underbrace{T^2 \cdot T^2 \cdot T^2 \cdots T^2}_{k/2} &\approx \underbrace{(XY) \cdot (XY) \cdot (XY) \cdots (XY)}_{k/2} \\ = X \cdot \underbrace{(YX) \cdot (YX) \cdots (YX)}_{k/2-1} \cdot Y &= X \cdot Z^{k/2-1} \cdot Y, \end{aligned} \tag{14}$$

where $Z_{c\times c} = Y_{c\times n} \cdot X_{n\times c}$. Therefore, we convert the computation of the k^{th} power of matrix T to the computation of the $(k/2-1)^{th}$ power of a small matrix Z.

We propose the T^2-Approx algorithm that uses the above idea to compute approximate PageRank values. Below, we convert the original equation of the PageRank (Equation 2) based on T^k into the formula based on $Z^{k/2-1}$, which guarantees the correctness of the T^2-Approx algorithm.

$$\begin{aligned}
\pi &= (1-\alpha)\cdot\left(\sum_{k=0}^{\infty}\alpha^k T^k\right)\cdot\pi_0 \approx (1-\alpha)\cdot\left(\sum_{k=0}^{\infty}\alpha^k (X\cdot Y)^{\frac{k}{2}}\right)\cdot\pi_0 \\
&= (1-\alpha)\pi_0 + (1-\alpha)\alpha\cdot\left(\sum_{k=1}^{\infty}(\alpha^2)^{\frac{k-1}{2}}\cdot X\cdot(Y\cdot X)^{\frac{k-1}{2}}\cdot Y\right)\cdot\pi_0 \\
&= (1-\alpha)\pi_0 + \frac{\alpha}{1+\alpha}\cdot X\cdot\left((1-\alpha^2)\cdot\sum_{k=1}^{\infty}(\alpha^2)^{\frac{k-1}{2}} Z^{\frac{k-1}{2}}\right) Y\cdot\pi_0 \\
&= (1-\alpha)\pi_0 + \frac{\alpha}{1+\alpha}\cdot X\cdot\underline{\left((1-\alpha')\cdot\sum_{k'=0}^{\infty}(\alpha')^{k'} Z^{k'}\cdot\tilde{\pi}_0\right)},
\end{aligned} \tag{15}$$

where $\alpha' = \alpha^2$, $k' = (k-1)/2$, and $\tilde{\pi}_0 = Y\cdot\pi_0$. The underlined part in the above equation can be computed using the power iteration method (Equation 3).

Algorithm 5.1 shows the pseudo code of the T^2-Approx algorithm. According to the given sampling probability distribution $\mathcal{F}$, matrix X is constructed by c sampled columns from T and matrix Y is constructed by taking the corresponding c rows from T (lines 1 and 2). Then, matrix Z is calculated as $Y\cdot X$ (line 3). Next, the initial PageRank vector is set to $\left[\frac{1}{n}\right]_n$ (line 4). According to Equation 15, the initial PageRank vector is first multiplied by matrix Y (line 5). After that, the power iteration method is conducted using matrix Z (lines 7–10). In the end, line 11 multiplies the current PageRank vector by matrix X and computes the approximate PageRank values according to Equation 15.

Algorithm 5.1 T^2-Approx

Input: The transition matrix T of graph G containing n vertices, the number of sampled columns and rows c, termination threshold τ, the damping factor α, the sampling probability distribution $\mathcal{F}$.

Output: Approximate PageRank values $\tilde{\pi}$ of G.

1: $X \leftarrow$ sampling c columns from T according to the probability distribution $\mathcal{F}$;
2: $Y \leftarrow$ taking the corresponding c rows from T;
3: $Z \leftarrow Y\cdot X$;
4: $\pi_0 \leftarrow \left[\frac{1}{n}\right]_n$;
5: $\tilde{\pi}^{(0)} \leftarrow Y\cdot\pi_0$;
6: $k \leftarrow 0$;
7: **repeat**
8: $\quad k \leftarrow k+1$;
9: $\quad \tilde{\pi}^{(k)} \leftarrow (1-\alpha^2)\cdot Z\cdot\tilde{\pi}^{(k-1)} + \alpha^2\left[\frac{1}{c}\right]_c$;
10: **until** $\|\tilde{\pi}^{(k)} - \tilde{\pi}^{(k-1)}\|_F < \tau$
11: $\tilde{\pi} \leftarrow (1-\alpha)\cdot\pi_0 + \frac{\alpha}{1+\alpha}\cdot X\cdot\tilde{\pi}^{(k)}$;
12: $\tilde{\pi} \leftarrow$ normalizing $\tilde{\pi}$;
13: **return** $\tilde{\pi}$;

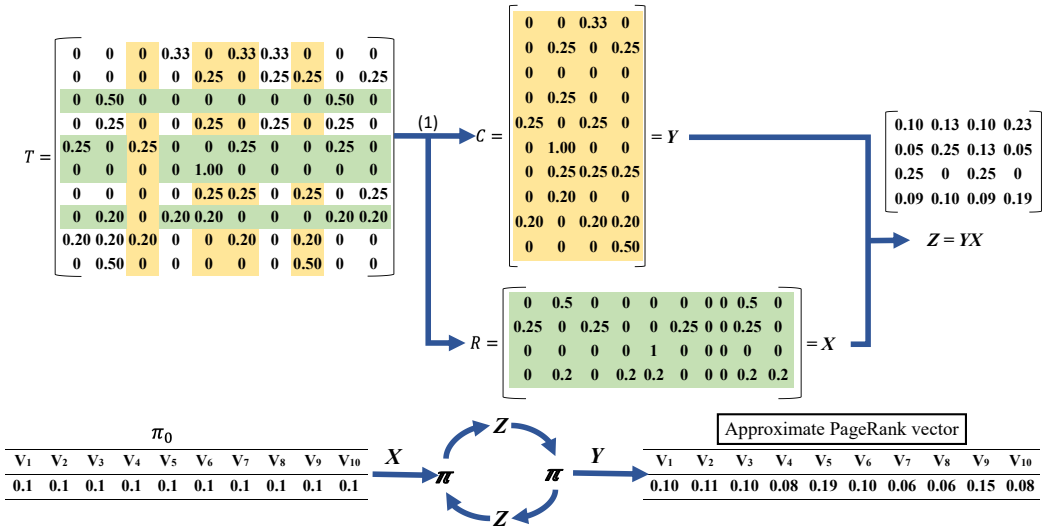


Fig. 4. T^2-Approx algorithm.

Example 2. Figure 4 illustrates the working process of algorithm T^2-Approx. Given a graph G, matrix T is shown in the figure. Suppose columns 3, 5, 6, and 8 in T are sampled and matrix Y is shown in the figure. Then, matrix X is composed of the corresponding rows, i.e, rows 3, 5, 6, and 8 in T. Next, matrix Z is calculated as $Y \cdot X$, as shown in the figure. After that, the initial PageRank vector π_0 is multiplied by matrix Y, resulting in vector $\tilde{\pi}$. Vector $\tilde{\pi}$ is iteratively multiplied by matrix Z until the termination threshold is satisfied. In the end, vector $\tilde{\pi}$ is multiplied by matrix X and returned as the approximate PageRank vector.

Discussion. The T^2-Approx algorithm uses modified MMC algorithm to approximate T^2 that has the following differences, compared with the original MMC algorithm.

(i) The modified MMC algorithm is independent of the sampling probability distribution, while traditional MMC uses the Euclidean-norm of column and row probability distribution (Equation 6)

(ii) For each sampled column and row, the traditional MMC algorithm performs a scaling operation that divides each element in the column and the row by $\sqrt{c \cdot p_i^c}$, which aims to guarantee an unbiased estimation of the matrix multiplication [21, 23]. The modified MMC algorithm does not perform this scaling operation, due to the same reason mentioned in Section 4.2.

Lemma 4. *The time complexity of the T^2-Approx algorithm is $O(c^2 \cdot n + c^2 \cdot (\log c + \log_\alpha \tau))$, where $c \ll n$.*

Proof. The computation cost of the T^2-Approx algorithm is dominated by two phases. In the first phase (lines 1–3), constructing matrices X, Y, and Z takes $O(c^2 \cdot n)$ time. In the second phase, the algorithm iteratively computes the PageRank vector using matrix Z (lines 7–10). Since the number k of iterations needed on matrix $T_{n\times n}$ is $O(\log n + \log_\alpha \tau)$ [63], the number of iterations needed on matrix $Z_{c\times c}$ is $O(\log c + \log_\alpha \tau)$. Given that the time complexity of the power iteration method on a graph having m edges is $O(k \cdot m)$ [20], we derive that the second phase takes $O(c^2 \cdot (\log c + \log_\alpha \tau))$ time, since the number of edges in the graph corresponding to matrix Z is bounded by c^2. Overall, the time complexity of T^2-Approx is $O(c^2 \cdot n + c^2 \cdot (\log c + \log_\alpha \tau))$, where $c \ll n$. □

5.3 Error Bound

Since we use the modified MMC algorithm in the T^2-Approx algorithm, the error bound of the MMC algorithm can not be applied. We derive the error bound of the T^2-Approx algorithm in Lemma 5.

Lemma 5. *According to algorithm T^2-Approx, we have*

$$\|T^2 - X \cdot Y\|_F \leq \sqrt{n-c}\|T\|_F^2.$$

The proof is available in the supplementary material.

Discussion. We proceed to compare the error bound of the T^2-Approx algorithm with that of existing matrix approximation-based algorithms listed in Table 1. According to Lemma 2, $\|T^k - (XY)^{k/2}\|_F \leq k/2\sqrt{n}\|T^2 - XY\|_F \leq k/2\sqrt{n(n-c)}\|T\|_F^2$. Considering the error bounds of DSPI and ASPI in Table 1, their bounds are based on $\|T\|_F^k$ that is larger than our bound based on $k \cdot \|T\|_F^2$. Based on Lemma 2, the error bound of the SVD-based PageRank approximation algorithm in Table 1 can be represented as $\|T^2 - \tilde{T}^2\| \leq 2\sqrt{n}\sqrt{\|T - T_\rho\|_F^2 + \varepsilon\|T\|_F^2}$, which is close to our bound.

6 Empirical Study

6.1 Experimental Setup

Datasets. We conduct evaluations on three real large graph datasets, which are summarized in Table 4. Friendster [57] and Orkut [58] are social networks from Friendster and Orkut websites, respectively. UKDomain [9] is a hyperlink network in 2007 in the United Kingdom.

Table 4. Statistics of datasets.

Dataset	# Vertices	# Edges
Orkut	3,072,441	117,184,899
Friendster	65,608,366	1,806,067,135
UKDomain	105,153,952	3,301,876,564

Competitors. The proposed algorithms CUR-Trans and T^2-Approx are compared with two representative sampling-based algorithms: the local PageRank algorithm with pruning (LPRAP) [1] and ApproxRank [76] and with one representative matrix approximation-based algorithm DSPI [51]. To further analyze the proposed transformation model, we include two variants of CUR-Trans that replace the CUR-based low-rank approximation in Algorithm 4.1 with the Truncated SVD algorithm [44] and the ConstantTime SVD algorithm [24], respectively.

The reasons for excluding other existing approximation PageRank algorithms are as follows. The iteration method and the Monte Carlo method involve the entire graph or the whole transition matrix in the iterative computation. The main memory consumption is very high and they are not applicable to very large graphs. Algorithm ASPI [51] computes a low-rank approximate matrix in each iteration that is time-consuming. The SVD-based PageRank approximation [7] involves the entire transition matrix in the SVD decomposition. It causes memory leaks on dataset Orkut. We have tested these algorithms on a small graph Gowalla [16] with 196,591 vertices. The iteration method terminates due to the memory leaks. The Monte Carlo-based algorithms [55, 66] spend more than 20 hours to perform just 6 iterations in distributed system and more than 2 hours on a single machine.

Metrics. We compare the proposed algorithms and the competitors in terms of the computation time and the error of the approximate results. Following the previous work [5, 36, 74], the ground

truth PageRank values are calculated by the power iteration algorithm for a given termination threshold τ. We use the implementation in the Python Networkit package [2] and set $\tau = \frac{1}{100 \cdot n}$ where n is the number of vertices in the graph. The computation time of the ground truth PageRank values on datasets Orkut, Friendster, and UKDomain are 11 seconds, 535 seconds, and 275 seconds, respectively. We calculate the absolute error between the ground truth PageRank value and the approximate PageRank value for each vertex, and then draw box plots.

Settings. Since the source code of the competitors are not available, we have implemented all the competitors. Specifically, algorithms DSPI and LPRAP compute PageRank values based on graph operations. They are implemented using Networkit (10.1) [2]. Algorithms ApproxRank, SVD-Trans, CUR-Trans, and T^2-Approx perform matrix iterations. They are implemented using NumPy (1.20.1) [32] and SciPy (1.8.1) [71]. All the algorithms are running on a machine with 4 Intel Xeon E7-4830 CPUs (56 cores, 2.0 GHz) and 2 TB main memory with Python (3.8.8).

Parameters. All the evaluated algorithms take the sampling ratio as a parameter. LPRAP and ApproxRank sample vertices in the graph. DSPI samples elements in the transition matrix. SVD-Trans, CUR-Trans, and T^2-Approx sample columns and rows in the transition matrix. To have a fair comparison, we tune the sampling ratios of different algorithms to generate the same amount of sampled edges. We report the performance of all the algorithms when the edge sampling ratio is set to 0.1%, 0.3%, 0.5%, 0.7%, and 1%. We did not use sampling ratios higher than 1%, because on very large graphs, 1% of the data is still a large graph. It is challenging for all the algorithms when using low sampling ratios.

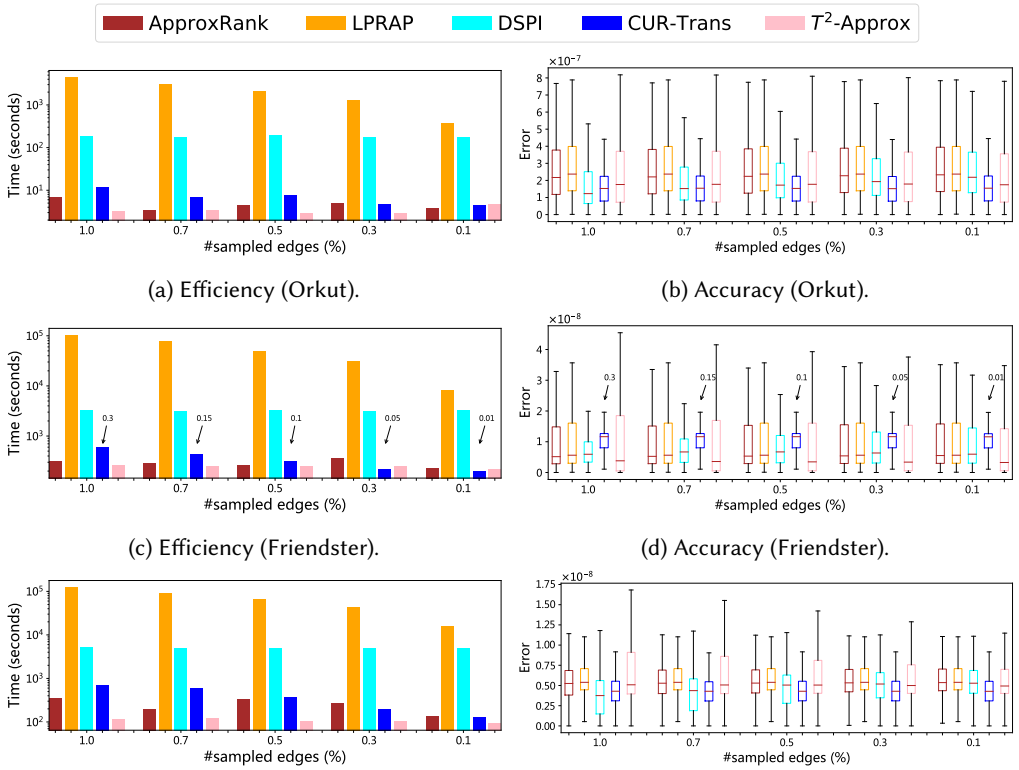
Algorithm LPRAP has a parameter that determines whether the neighbors of a sampled vertex will be included in the next iteration. It has been tuned and set to 10^{-8}, 10^{-9}, and 10^{-10} on datasets Orkut, Friendster, and UKDomain, respectively. In algorithm DSPI, parameter θ is used to calculate the sampling probability. We tune θ to get the best results of DSPI on the three datasets.

6.2 Experimental Results.

Main results. Figures 5a, 5c, and 5e compare the computation time of difference algorithms. Figures 5b, 5d, and 5f present the box plot of the errors of the approximate PageRank values returned by different algorithms. Figure 5g includes the amount of vertices with approximate PageRank values provided by different algorithms. Our algorithms CUR-Trans and T^2-Approx report approximate PageRank values for all the vertices in the graph, while the two sampling-based algorithms ApproxRank and LPRAP only estimate PageRank values for sampled vertices that is a small portion of the graph. Algorithm DSPI approximates the transition matrix by conducting element-wise sampling. It may generate isolated vertices, and thus the PageRank values of these isolated vertices cannot be estimated. Since the competitor algorithms cannot estimate PageRank values for all the vertices, we use 0 as the PageRank values of these vertices when computing the errors.

Algorithms DSPI and LPRAP are significantly slower than the other algorithms. Because DSPI conducts element-wise sampling in the transition matrix that is equivalent to traverse the entire edge set in the graph. LPRAP needs to perform graph exploration to obtain the local graph structures for sampled vertices, which is expensive. Algorithm ApproxRank is not slow, since it only computes PageRank values for less than 10% of the vertices. Our algorithms CUR-Trans and T^2-Approx are fast on all the datasets when using different sampling ratios. T^2-Approx is faster than CUR-Trans, which proves that the time complexity of T^2-Approx is lower.

The errors of DSPI and T^2-Approx are comparable on datasets Orkut. The error of T^2-Approx is better than that of DSPI on dataset Friendster, but slightly worse than that of DSPI on dataset UKDomain. ApproxRank and LPRAP provide good estimations on Orkut and UKDomain, but perform bad on UKDomain. CUR-Trans is able to offer good estimations on all the datasets when



(a) Efficiency (Orkut).

(b) Accuracy (Orkut).

(c) Efficiency (Friendster).

(d) Accuracy (Friendster).

(e) Efficiency (UKDomain).

(f) Accuracy (UKDomain).

	ApproxRank	LPRAP	DSPI	CUR-Trans	T^2-Approx
Orkut	2.00%–9.00%	0.03%	8.90%–59.19%	100%	100%
Friendster	2.00%–9.00%	0.002%	12.68%–64.91%	100%	100%
UKDomain	2.00%–9.00%	0.001%	13.29%–55.32%	100%	100%

(g) The amount of returned vertices with PageRank values.

Fig. 5. Performance of different approximate PageRank algorithms.

using different sampling ratios. All the competitor algorithms only report the PageRank values of a small amount of vertices, while our algorithms CUR-Trans and T^2-Approx can provide estimated results for all the vertices.

To sum up, our algorithms CUR-Trans and T^2-Approx are able to provide good estimations on the PageRank values of all the vertices when using small sampling ratios. They outperform the competitors in terms of the computation time. The error of CUR-Trans is better than that of T^2-Approx, while T^2-Approx is more efficient than CUR-Trans.

Effect of the edge sampling ratio. We analyze the performance of all the algorithms when the edge sampling ratio varies from 0.1% to 1%, shown in Figure 5. Different from the other algorithms, the edge sampling ratio of CUR-Trans varies from 0.01% to 0.3% on dataset Friendster as shown in Figures 5c and 5d. The reason is that when using the edge sampling ratio larger than 0.3%, the

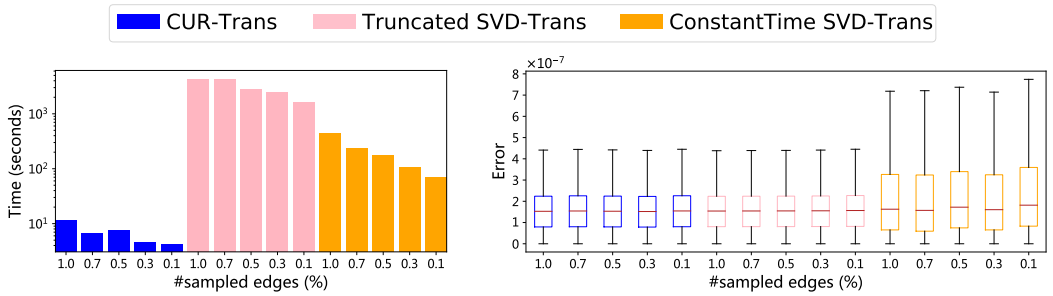
main memory size is not large enough for decomposing the matrix constructed by the sampled rows and columns on dataset Friendster.

The computation time of LPRAP increases as the edge sampling ratio increases, since more edges are involved in the computation. However, the error of LPRAP does not change much with the edge sampling ratio. It is because that LPRAP traverses the incident edges of the sampled vertices and the error of LPRAP is affected by the number of traversed hops away from the sampled vertices. When the edge sampling ratio varies from 0.1% to 1%, the number of traversed hops does not increase much, so that the error of LPRAP is not improved. The errors of DSPI are improved as the edge sampling ratio increases, which is expected. However, the computation time does not change much. The reason is that DSPI generates samples by traversing all the elements in the adjacency matrix, which dominates the computation cost. Thus, the edge sampling ratio does not affect the computation time much. There is no obvious trend of high edge sampling ratio yielding more computation time of ApproxRank. It is because that the computation time of ApproxRank is determined by the size of the adjacency matrix of the sampled subgraph. Given an edge sampling ratio, we may obtain less vertices when high-degree vertices are chosen and we may obtain more vertices when low-degree vertices are chosen. Due to the randomness of the sampling process, the computation time of ApproxRank is not correlated with the edge sampling ratio. Since ApproxRank only estimates PageRank values for less than 10% of the vertices in the graph and the rest of the vertices are assigned 0 values, the error does not change much with the edge sampling ratio. The computation time of CUR-Trans increases as the edge sampling ratio increases, which is expected. The error does not change much. It is because that in the CUR decomposition, when the numbers of sampled columns and rows exceed a certain multiple of the rank of the matrix, the error becomes stable. In most cases, the computation time of T^2-Approx increases as the edge sampling ratio increases, which is expected. It spends a little bit more time when the sampling ratio is 0.1% on dataset Orkut. The reason is that the computation on a small graph may require more number of iterations to converge. The error of T^2-Approx is not sensitive to the edge sampling ratio.

CUR-Trans vs. SVD-Trans. We compare the performance of CUR-Trans with two SVD-based variants, i.e., Truncated SVD-Trans and ConstantTime SVD-Trans, in Figure 6. It is observed that the errors of algorithms CUR-Trans and Truncated SVD-Trans are comparable, because both CUR-based low-rank approximation and Truncated SVD-based low-rank approximation can preserve the information in the transition matrix. The error of ConstantTime SVD-Trans is worse than the other two algorithms, since ConstantTime SVD gains efficiency by sacrificing the accuracy of the matrix approximation. CUR-Trans is significantly faster than both Truncated SVD-Trans and ConstantTime SVD-Trans, since the modified CUR decomposition in the CUR-Trans algorithm is efficient. Later, if better low-rank approximation algorithms are developed, it is easy to incorporate them into our transformation model for PageRank approximation.

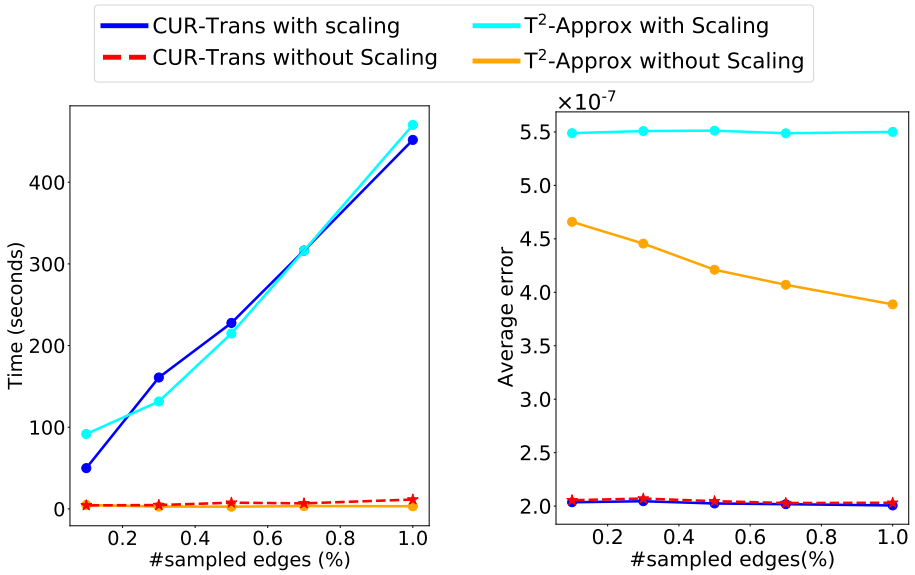
Effect of scaling operation. Sections 4.2 and 5.2 explain the reasons why the proposed algorithms CUR-Trans and T^2-Approx do not use the scaling operation. This experiment studies how the scaling operation affects the performance. Figure 7 compares the computation time and the average errors of the proposed algorithms CUR-Trans and T^2-Approx (without scaling operations) and the variants of CUR-Trans and T^2-Approx with scaling operations. Figure 7a shows that both CUR-Trans and T^2-Approx without scaling operations are significant faster than the corresponding versions with scaling operations. Figure 7b shows that the average error of CUR-Trans with scaling operation is not better than the proposed CUR-Trans and the average error of T^2-Approx with scaling operation is worse than the proposed T^2-Approx.

Effect of the rank-x approximation in the CUR-Trans algorithm. In the CUR-Trans algorithm, the modified CUR decomposition constructs a rank-x approximation of U. Figure 8 shows the computation time and the error of the CUR-Trans algorithm when varying x on dataset Orkut. We



(a) Efficiency. (b) Accuracy.

Fig. 6. CUR-Trans vs. SVD-Trans (Orkut).

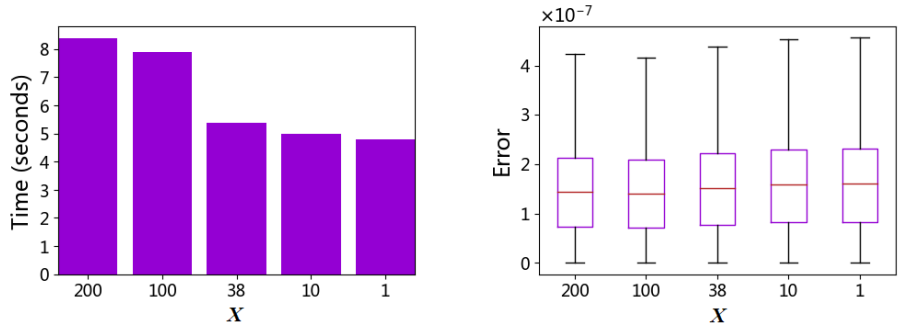


(a) Efficiency. (b) Accuracy.

Fig. 7. Effect of scaling operation (Orkut).

observe that the error increases slightly as x decreases, meaning that given different values of x, the generated low-rank approximation matrices $C \cdot \tilde{U}_x \cdot R$ are similar. The computation time of the CUR-Trans algorithm decreases as x decreases, meaning that choosing a lower-rank approximation saves the computation cost. According to the CUR-Trans algorithm, on dataset Orkut, setting $x = \sqrt{c} = 38$ can gain large efficiency improvement while keeping the error low.

Comparison with open source systems. We compare the performance of our algorithms with open source systems: Networkit [67], GraphX [79], and Giraph [37] in Table 5. GraphX and Giraph are distributed systems. Networkit is a single-machine system. The edge sampling ratio of our algorithms CUR-Trans and T^2-Approx is set to 0.1%. Our algorithms are faster than the open source systems. GraphX and Giraph are slow due to the high communication cost. Networkit is faster than GraphX and Giraph, but requires large-size main memory on a single machine. Our algorithms



(a) Efficiency. (b) Accuracy.

Fig. 8. Performance of CUR-Trans when varying x (Orkut).

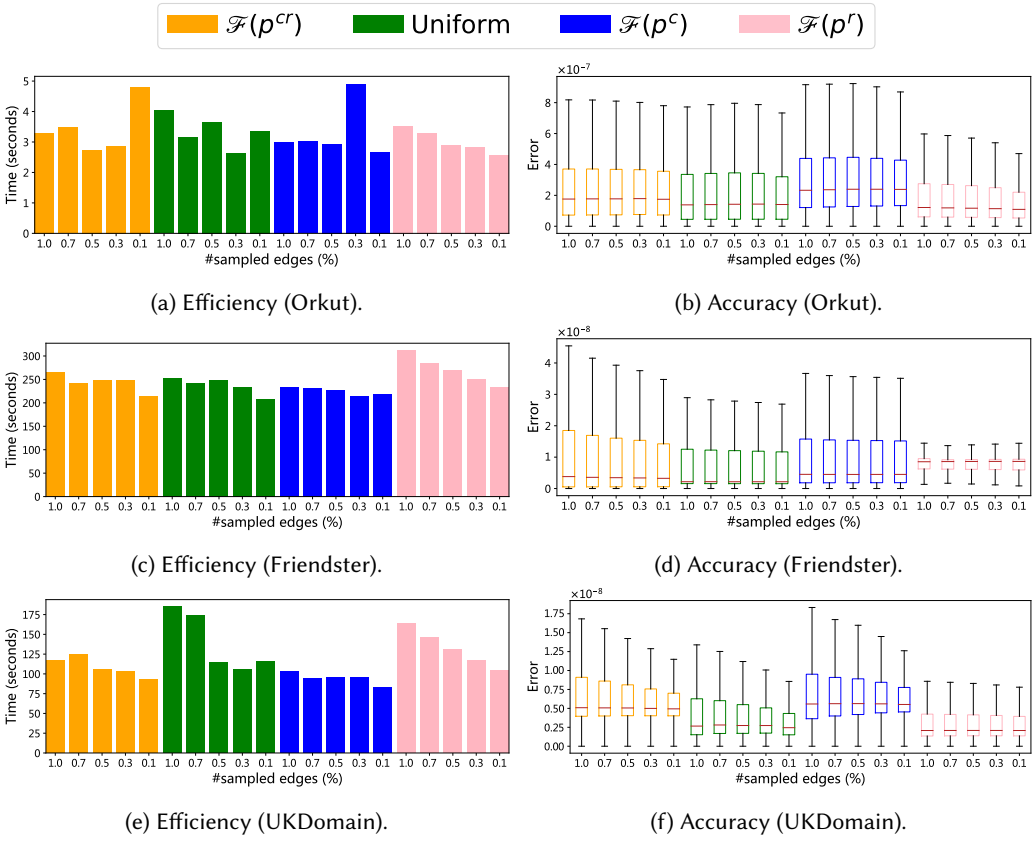
Table 5. Our algorithms vs. open source systems.

	Orkut	Friendster	UK2007
Networkit	11 seconds	535 seconds	275 seconds
GraphX	46 minutes	3+ hours	3+ hours
Giraph	11 minutes	3+ hours	3+ hours
CUR-Trans (0.1%)	4 seconds	198 seconds	129 seconds
T^2-Approx (0.1%)	5 seconds	215 seconds	93 seconds

convert the PageRank computation on a large graph to a small graph, which require less main memory and have low computation cost.

Effect of the sampling probability distribution in the T^2-Approx algorithm. According to Lemma 5, the error bound of the T^2-Approx algorithm is independent of the sampling probability distribution. To justify this result, Figure 9 compares the computation time and the errors of the T^2-Approx algorithm using four different sampling probability distributions, i.e., the uniform distribution and the three Euclidean norm based sampling probability distributions (Equations 4, 5, and 6) described in Section 3. The experiment results show that the computation time of the four sampling probability distributions are comparable. The error of the Euclidean norm of column probability distribution (Equation 4) is slightly worse than that of the other distributions on datasets Orkut and UKDomain. The error of the Euclidean norm of row probability distribution (Equation 5) is slightly worse than that of the other distributions on dataset Friendster. Hence, it is free to use different sampling probability distributions in the T^2-Approx algorithm.

Comparison of the vertex ranking. We rank the vertices in the descending order of their ground truth PageRank values and take this ranking as the ideal ranking. Next, for each algorithm, we obtain its vertex ranking in the descending order of their estimated PageRank values. Then, we compute the Normalized Discounted Cumulative Gain (NDCG) [38] of the vertex ranking produced by each algorithm. A high NDCG value indicates that the vertex ranking of an algorithm is close to the ideal ranking. Figure 10 shows the NDCG values of all the algorithms on dataset Orkut. DSPI performs the best. CUR-Trans is slightly worse than DSPI, but it is significantly faster than DSPI as shown in Figure 5. CUR-Trans achieves higher NDCG values compared with ApproxRank and LPRAP. Although T^2-Approx is a little bit worse than the other algorithms, it is significantly faster than the competitor algorithms as shown in Figure 5.



(a) Efficiency (Orkut).

(b) Accuracy (Orkut).

(c) Efficiency (Friendster).

(d) Accuracy (Friendster).

(e) Efficiency (UKDomain).

(f) Accuracy (UKDomain).

Fig. 9. Performance of the T^2-Approx algorithm using different sampling probability distributions.

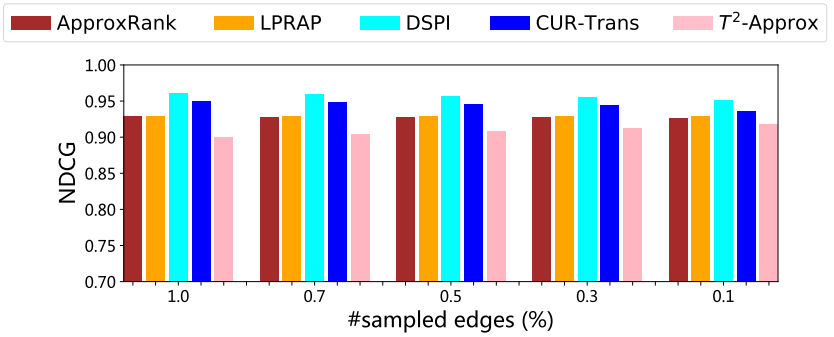


Fig. 10. NDCG results of different algorithms (Orkut).

7 Conclusion

PageRank is an important measurement, which has been widely used in various applications (e.g., search engines, recommendation systems, and social networks). Despite extensive efforts to develop efficient and accurate algorithms for computing PageRank values, existing methods

cannot efficiently provide accurate estimations for this measurement. To tackle this issue, we first propose the CUR-Trans algorithm, which can reduce the time complexity for approximately computing PageRank values with lower error bounds. Then, we propose the T^2-Approx algorithm to further reduce the time complexity for handling this operation. Experiment results show that both CUR-Trans and T^2-Approx algorithms can achieve the lowest response time for computing PageRank on large graphs with the best accuracy (for the CUR-Trans algorithm) or competitive accuracy (for the T^2-Approx algorithm).

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants 62372308 and 62202401 and the GuangDong Basic and Applied Basic Research Foundation under Grant 2023A1515011619.

References

[1] Ziv Bar-Yossef andLi-Tal Mashiach. 2008. Local approximation of pagerank and reverse pagerank. In *CIKM*. 279–288.
[2] Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck. 2022. Algorithms for Large-Scale Network Analysis and the NetworKit Toolkit. In *Algorithms for Big Data - DFG Priority Program 1736*. Vol. 13201. 3–20.
[3] Anton Anikin, Alexander Gasnikov, Alexander Gornov, Dmitry Kamzolov, Yury Maximov, and Yurii Nesterov. 2022. Efficient numerical methods to solve sparse linear equations with application to pagerank. *Optimization Methods and Software* 37 (2022), 907–935.
[4] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient. *SIAM J. Numer. Anal.* 45 (2007), 890–904.
[5] Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. 2011. Fast personalized PageRank on MapReduce. In *SIGMOD*. 973–984.
[6] Sudipto Banerjee and Anindya Roy. 2014. *Linear algebra and matrix analysis for statistics*. Crc Press.
[7] András A. Benczúr, Károly Csalogány, and Tamás Sarlós. 2005. On the feasibility of low-rank approximation for personalized PageRank. In *WWW*. 972–973.
[8] Monica Bianchini, Marco Gori, and Franco Scarselli. 2005. Inside PageRank. *ACM Trans. Internet Techn.* 5 (2005), 92–128.
[9] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. UbiCrawler: a scalable fully distributed Web crawler. *Softw. Pract. Exp.* 34 (2004), 711–726.
[10] Arsineh Boodaghian Asl, Jayanth Raghothama, Adam Darwich, and Sebastiaan Meijer. 2021. using PageRank and social network analysis to sprcify mental health factors. *Proceedings of the Design Society* 1 (2021), 3379–3388.
[11] L. A. Breyer. 2002. *Markovian page ranking distributions: some theory and simulations*. Technical Report.
[12] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Networks* 30, 1-7 (1998), 107–117.
[13] Jie Chen and Yousef Saad. 2009. On the tensor SVD and the optimal low rank orthogonal approximation of tensors. *SIAM journal on Matrix Analysis and Applications* 30 (2009), 1709–1734.
[14] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. 2004. Local methods for estimating pagerank values. In *CIKM*. 381–389.
[15] Zhen Chen, Xingzhi Guo, Baojian Zhou, Deqing Yang, and Steven Skiena. 2023. Accelerating Personalized PageRank Vector Computation. In *KDD*. 262–273.
[16] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*. 1082–1090.
[17] Fan Chung. 2014. A Brief Survey of PageRank Algorithms. *IEEE Trans. Netw. Sci. Eng.* 1 (2014), 38–42.
[18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.
[19] Gianna M. Del Corso, Antonio Gulli, and Francesco Romani. 2005. Fast PageRank Computation via a Sparse Linear System. *Internet Math.* 2 (2005), 251–273.
[20] Jason V. Davis and Inderjit S. Dhillon. 2006. Estimating the global pagerank of web communities. In *SIGKDD*. 116–125.
[21] Petros Drineas and Ravi Kannan. 2001. Fast Monte-Carlo Algorithms for Approximate Matrix Multiplication. In *FOCS*. 452–459.
[22] Petros Drineas and Ravi Kannan. 2003. Pass efficient algorithms for approximating large matrices. In *ACM-SIAM*. 223–232.

[23] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. 2006. Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication. *SIAM J. Comput.* 36 (2006), 132–157.
[24] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. 2006. Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix. *SIAM J. Comput.* 36 (2006), 158–183.
[25] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. 2006. Fast Monte Carlo Algorithms for Matrices III: Computing a Compressed Approximate Matrix Decomposition. *SIAM J. Comput.* 36 (2006), 184–206.
[26] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. 2008. Relative-Error CUR Matrix Decompositions. *SIAM J. Matrix Anal. Appl.* 30 (2008), 844–881.
[27] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1 (1936), 211–218.
[28] David F Gleich, Andrew P Gray, Chen Greif, and Tracy Lau. 2010. An inner-outer iteration for computing PageRank. *SIAM Journal on Scientific Computing* 32 (2010), 349–371.
[29] Gene H Golub, Alan Hoffman, and Gilbert W Stewart. 1987. A generalization of the Eckart-Young-Mirsky matrix approximation theorem. *Linear Algebra and its applications* 88 (1987), 317–327.
[30] Gene H Golub and Christian Reinsch. 1971. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation: Volume II: Linear Algebra*. Springer, 134–151.
[31] Chuanqing Gu, Fei Xie, and Ke Zhang. 2015. A two-step matrix splitting iteration for computing PageRank. *J. Comput. Appl. Math.* 278 (2015), 19–28.
[32] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nat.* 585 (2020), 357–362.
[33] Somaia Awad Hassan, A. M. Hemeida, and Mountasser M. M. Mahmoud. 2016. Performance Evaluation of Matrix-Matrix Multiplications Using Intel's Advanced Vector Extensions (AVX). *Microprocess. Microsystems* 47 (2016), 369–374.
[34] Taher Haveliwala et al. 1999. *Efficient computation of PageRank*. Technical Report. Citeseer.
[35] Fumio Hiai and Dénes Petz. 2014. *Introduction to matrix analysis and applications*. Springer Science & Business Media.
[36] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively Parallel Algorithms for Personalized PageRank. *Proc. VLDB Endow.* 14 (2021), 1668–1680.
[37] J Jackson. 2013. Facebook's graph search puts Apache Giraph on the map. *Retrieved July* 25 (2013), 2015.
[38] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
[39] Sepandar Kamvar, Taher Haveliwala, and Gene Golub. 2004. Adaptive methods for the computation of PageRank. *Linear Algebra Appl.* 386 (2004), 51–65.
[40] Sepandar D Kamvar, Taher H Haveliwala, Christopher Manning, and Gene H Golub. 2003. *Exploiting the block structure of the web for computing pagerank*. Technical Report.
[41] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. 2003. Extrapolation methods for accelerating PageRank computations. In *WWW*. 261–270.
[42] Seunghwa Kang, Joseph Nke, and Brad Rees. 2022. Analyzing Multi-trillion Edge Graphs on Large GPU Clusters: A Case Study with PageRank. In *HPEC*. 1–7.
[43] George Karypis and Vipin Kumar. 1999. Parallel Multilevel series k-Way Partitioning Scheme for Irregular Graphs. *SIAM Rev.* 41 (1999), 278–300.
[44] N Kishore Kumar and Jan Schneider. 2017. Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra* 65, 11 (2017), 2212–2244.
[45] Miloš Kotlar, Marija Punt, and Veljko Milutinović. 2022. Energy efficient implementation of tensor operations using dataflow paradigm for machine learning. In *Advances in Computers*. Vol. 126. 151–199.
[46] Liang Lan, Kai Zhang, Hancheng Ge, Wei Cheng, Jun Liu, Andreas Rauber, Xiao-Li Li, Jun Wang, and Hongyuan Zha. 2017. Low-rank decomposition meets kernel learning: A generalized Nyström method. *Artificial Intelligence* 250 (2017), 1–15.
[47] Amy Nicole Langville and Carl Dean Meyer. 2003. Survey: Deeper Inside PageRank. *Internet Math.* 1 (2003), 335–380.
[48] Mu Li, James Tin-Yau Kwok, and Baoliang Lü. 2010. Making large-scale Nyström approximation possible. In *ICML*. 631.
[49] Xiaocan Li, Shuo Wang, and Yinghao Cai. 2019. Tutorial: Complexity analysis of singular value decomposition and its variants. *arXiv* (2019).
[50] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *SIGMOD*. 2048–2061.

[51] Wenting Liu, Guangxia Li, and James Cheng. 2015. Fast PageRank approximation by adaptive sampling. *Knowl. Inf. Syst.* 42 (2015), 127–146.

[52] Jun Lu. 2021. A rigorous introduction to linear models. *arXiv* (2021).

[53] Jun Lu. 2022. Matrix decomposition and applications. *arXiv* (2022).

[54] Siqiang Luo. 2019. Distributed PageRank Computation: An Improved Theoretical Study. In *AAAI*. 4496–4503.

[55] Siqiang Luo, Xiaowei Wu, and Ben Kao. 2022. Distributed PageRank computation with improved round complexities. *Inf. Sci.* 607 (2022), 109–125.

[56] Michael W. Mahoney and Petros Drineas. 2009. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA* 106, 3 (2009), 697–702.

[57] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *NIPS*. 548–556.

[58] Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *SIGCOMM*. 29–42.

[59] Dingheng Mo and Siqiang Luo. 2023. Single-Source Personalized PageRanks With Workload Robustness. *IEEE Trans. Knowl. Data Eng.* 35 (2023), 6320–6334.

[60] Yuji Nakatsukasa. 2020. Fast and stable randomized low-rank matrix approximation. *arXiv:2009.11392* (2020).

[61] José Ignacio Orlicki, Pablo Ignacio Fierens, and J. Ignacio Alvarez-Hamelin. 2008. Faceted Ranking of Egos in Collaborative Tagging Systems. *CoRR* abs/0809.4668 (2008).

[62] José Ignacio Orlicki, Pablo Ignacio Fierens, and J. Ignacio Alvarez-Hamelin. 2009. Faceted Ranking in Collaborative Tagging Systems - Efficient Algorithms for Ranking Users based on a Set of Tags. In *WEBIST*. 626–633.

[63] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1998. *The pagerank citation ranking: Bring order to the web.* Technical Report. Technical report, stanford University.

[64] C-T Pan. 2000. On the existence and computation of rank-revealing LU factorizations. *Linear Algebra Appl.* 316 (2000), 199–222.

[65] R Piziak and PL Odell. 1999. Full rank factorization of matrices. *Mathematics magazine* 72 (1999), 193–201.

[66] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2015. Fast distributed PageRank computation. *Theor. Comput. Sci.* 561 (2015), 113–121.

[67] Christian L Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Network Science* 4, 4 (2016), 508–530.

[68] Min Tao, Xinmin Yang, Gao Gu, and Bohan Li. 2020. Paper recommend based on LDA and PageRank. In *ICAIS*. 571–584.

[69] Zhaolu Tian, Yong Liu, Yan Zhang, Zhongyun Liu, and Maoyi Tian. 2019. The general inner-outer iteration method based on regular splittings for the PageRank problem. *Appl. Math. Comput.* 356 (2019), 479–501.

[70] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *ICDM*. 613–622.

[71] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.

[72] Hanzhi Wang, Zhewei Wei, Junhao Gan, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2022. Edge-based Local Push for Personalized PageRank. *Proc. VLDB Endow.* 15 (2022), 1376–1389.

[73] Sibo Wang and Yufei Tao. 2018. Efficient Algorithms for Finding Approximate Heavy Hitters in Personalized PageRanks. In *SIGMOD*. 1113–1127.

[74] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *ACM Trans. Database Syst.* 44 (2019), 18:1–18:37.

[75] Yizhuo Wang, Weixing Ji, Xu Chen, and Sensen Hu. 2015. Task Parallel Implementation of Matrix Multiplication on Multi-socket Multi-core Architectures. In *ICA3PP*, Vol. 9530. 93–104.

[76] Yao Wu and Louiqa Raschid. 2009. ApproxRank: Estimating Rank for a Subgraph. In *ICDE*. 54–65.

[77] Yajun Xie, Lihua Hu, and Changfeng Ma. 2023. A Parameterized Multi-Splitting Iterative Method for Solving the PageRank Problem. *Mathematics* 11, 15 (2023), 3320.

[78] Ya-Jun Xie and Chang-Feng Ma. 2018. A relaxed two-step splitting iteration method for computing PageRank. *Computational and Applied Mathematics* 37 (2018), 221–233.

[79] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. 2013. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*. 1–6.

[80] Jieping Ye and Qi Li. 2005. A two-stage linear discriminant analysis via QR-decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2005), 929–941.
[81] Fuzhen Zhang. 2015. A matrix decomposition and its applications. *Linear and Multilinear Algebra* 63 (2015), 2033–2042.
[82] Shijie Zhou, Kartik Lakhotia, Shreyas G. Singapura, Hanqing Zeng, Rajgopal Kannan, Viktor K. Prasanna, James Fox, Euna Kim, Oded Green, and David A. Bader. 2017. Design and implementation of parallel PageRank on multicore platforms. In *HPEC*. 1–6.

Received January 2024; revised April 2024; accepted May 2024