PJsim: Towards Precise and Scalable Graph Similarity

PRAJJWAL NIJHARA*, Indian Institute of Technology Jodhpur, India
JAINAN TANDEL, Indian Institute of Technology Jodhpur, India
ROHIT PRAJAPATI, Indian Institute of Technology Jodhpur, India
DIP SANKAR BANERJEE, Indian Institute of Technology Jodhpur, India

Quantifying node similarity is central to graph analysis, particularly in social networks. SimRank, a popular measure by Jeh and Widom [19], defines similarity recursively: two nodes are similar if they are referenced by similar nodes. However, its limitation to equal-length paths leads to the "zero-similarity" problem, where structurally related nodes may score zero due to the lack of symmetric in-neighbor paths, especially in sparse or hierarchical graphs. While several variants of SimRank have been proposed to overcome this problem, most rely on iterative methods that approximate the solution by exploring paths of varying lengths. These approaches often require expensive convergence checks and can struggle with scalability on large graphs. To address these limitations, we propose PJsim[1], a novel, exact, and closed-form solution for SimRank computation. PJsim reformulates the recursive similarity definition as Sylvester equations and solves them via block transformations. Our experiments on real-world and synthetic graphs demonstrate that PJsim consistently delivers higher accuracy than traditional methods and runs up to 3.22× faster on synthetic datasets and 2.15× faster on real-world graphs, compared to state-of-the-art solutions addressing the zero-similarity problem.

CCS Concepts: • **Information systems** → **Similarity measures**; • **Mathematics of computing** → *Graph algorithms.*

Additional Key Words and Phrases: Edge Ranking, SimRank, Zero-similarity problem, Non-iterative algorithms, Closed-form solution, Semantic fidelity

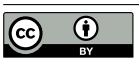
ACM Reference Format:
Prajjwal Nijhara, Jainan Tandel, Rohit Prajapati, and Dip Sankar Banerjee. 2026. PJsim: Towards Precise and Scalable Graph Similarity. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 223 (June 2026), 25 pages. https://doi.org/10.1145/3802100

1 Introduction

The concept of **link-based similarity** in graph theory is a key tool for identifying relationships between nodes in many applications, including recommendation systems, hyperlink analysis, and spam detection [3, 19, 38], where graphs naturally represent entities and their connections. Similarity in a structural sense can be understood as: two nodes are considered similar if they share common neighbors or exhibit analogous neighborhood structures, even when they are not directly connected by an edge. However, the irregular and complex structure of real-world networks poses challenges for measuring node similarity [29]. As a result, building reliable and scalable link-based

*Prajjwal Nijhara is the corresponding author.
[1]PJsim is named after the first letters of the authors' names.

Authors' Contact Information: Prajjwal Nijhara, Indian Institute of Technology Jodhpur, Karwar, Rajasthan, 342030, India, d23cse005@iitj.ac.in; Jainan Tandel, Indian Institute of Technology Jodhpur, Karwar, Rajasthan, 342030, India, m23csa010@alumni.iitj.ac.in; Rohit Prajapati, Indian Institute of Technology Jodhpur, Karwar, Rajasthan, 342030, India, p24cs0201@iitj.ac.in; Dip Sankar Banerjee, Indian Institute of Technology Jodhpur, Karwar, Rajasthan, 342030, India, dipsankarb@iitj.ac.in.



ACM 2836-6573/2026/6-ART223
https://doi.org/10.1145/3802100

similarity models has become an important goal. These models should not only capture *basic notions of similarity through clear mathematical definitions* derived from graph topology rather than explicit node attributes, but also **scale efficiently to large datasets** to support real-world use.

Jeh and Widom introduced the **SimRank algorithm (S_{Jeh})** in 2002 as a simple yet effective approach to measuring link-based similarity [19]. The core principle of S_{Jeh} is "*two nodes are similar if similar nodes point to them,*" which quickly gathered significant attention. Initially designed for directed graphs, its applicability has since been extended to undirected graphs as well. The simple and intuitive nature of SimRank has led to its widespread adoption across various fields, including social network analysis [46], k-nearest neighbor searches [22], clustering [6], query rewriting [2], and citation and hyperlink analysis [32, 35]. Its ability to capture **structural similarity** has established S_{Jeh} as a valuable tool in numerous graph-based applications within both research and practical settings.

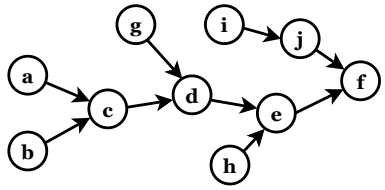
Building upon the success of S_{Jeh}, subsequent research has aimed to **reduce its high computational overhead** and **improve scalability**. Li et al. proposed a fast variant of S_{Jeh} applicable to both static and dynamic graphs [25]. A key distinction between Li's model and S_{Jeh} lies in the *treatment of self-similarity*: while Jeh and Widom fix the self-similarity of a node to 1, Li's formulation relaxes this constraint to allow more flexibility. However, both approaches retain an iterative structure with a time complexity of $O(K \cdot n^2 \cdot d^2)$ (where n is the number of nodes, d is the average in-degree, and K is the number of iterations), rendering them impractical for large-scale graphs.

The computational overhead of S_{Jeh} has led to the development of several **fast and approximate algorithms** [10, 11, 17, 26, 41, 43]. A key challenge that remains is the *zero-similarity problem*, which refers to the assignment of a similarity score of exactly zero to a node pair despite the presence of structural evidence (e.g., multi-hop connectivity). This arises from the parity restriction of the recursive SimRank formulation rather than from an absence of a meaningful relationship. Traditional methods often rely on *equal-length paths from common neighbors*, ignoring other valid connections [14, 40, 42]. This restriction creates a **parity effect**, the phenomenon in S_{Jeh} where node pairs connected only through odd-length paths receive zero-similarity scores. As a result, two nodes may receive a similarity score of zero even when they are structurally related through **multi-hop structural relationships**, defined as the propagation of similarity through sequences of shared neighbors, where nodes connected through multi-step paths would be expected to exhibit non-zero-similarity. This limitation is especially pronounced in graphs where **node similarity does not imply direct linkage**, such as sparse graphs and recommendation systems. The issue appears clearly in the original S_{Jeh} model, which requires shared in-neighbors to propagate similarity. As shown in Figure 1, when computing similarity to node f, S_{Jeh} assigns a score of zero to all other nodes because none share a direct in-neighbor with f. Nodes like a, b, and j, which lie on paths leading to f, are excluded from the similarity computation. *This restriction limits the model's ability to capture meaningful patterns that depend on indirect structure.*

SimRank* (S_{Yu}), proposed by Yu et al. [42], addresses this issue by generalizing S_{Jeh}'s recursive formulation to allow **asymmetric influence propagation** and **path-length diversity**. In the same example, S_{Yu} assigns *smoothly decaying non-zero-similarity scores* to all structurally relevant nodes, better reflecting the underlying topology. While S_{Yu} represents the state-of-the-art solution for addressing the zero-similarity problem, its work complexity is $O(Kmn)$. For large and dense graphs, this complexity essentially becomes $O(Kn^3)$.

1.1 Motivation and Contributions

While S_{Yu} effectively improves accuracy by incorporating more comprehensive path structures, its iterative formulation inherits the computational drawbacks of earlier models. Since S_{Yu} is derived from the iterative definition of S_{Jeh}, it relies on a convergence-based iterative method. To find the



Pairs	Similarity $(\star, f)$	
$(\star, f)$	S_{Jeh}	S_{Yu}
(a, f)	0.0000	0.0051
(b, f)	0.0000	0.0051
(c, f)	0.0000	0.0333
(d, f)	0.0000	0.0975
(e, f)	0.0000	0.2207
(f, f)	1.0000	0.4713
(g, f)	0.0000	0.0128
(h, f)	0.0000	0.0320
(i, f)	0.0000	0.0320
(j, f)	0.0000	0.1184

Fig. 1. A zero-similarity problem on a citation graph when similarities $s(\star, f)$ w.r.t. query f are assessed. S_{Jeh} assigns zero to all non-identical nodes due to the lack of shared in-neighbors with f, while S_{Yu} assigns non-zero scores based on indirect structural relationships.

Table 1. Iteration counts and total runtime of S_{Yu} for all-pairs similarity scores under different tolerance values, illustrating how the approximation level affects convergence and computation time.

Graph	Nodes	Edges	Tolerance	Iterations	Time (s)
Wikipedia	4.7K	185.4K	10^{-2}	29	49.10
			10^{-4}	32	54.20
			10^{-6}	35	59.25
Gnutella04	10.8K	39.9K	10^{-2}	9	501.17
			10^{-4}	24	1189.08
			10^{-6}	38	1972.62
Twitter	81.3K	1.76M	10^{-2}	7	11747.36
			10^{-4}	22	33621.09
			10^{-6}	41	60948.60
Orkut	117.2K	1.08M	10^{-2}	6	4835.77
			10^{-4}	18	9984.35
			10^{-6}	33	17253.89

similarity scores, the equations must converge, which requires multiple iterations. These iterations can vary from tens to a few hundred, depending on the graph structure. The accuracy of results can be controlled by either limiting the number of iterations or by defining a **tolerance value** [13, 28], a user-defined threshold that specifies the acceptable approximation level in iterative SimRank methods; the iteration stops once the Frobenius-norm difference between successive similarity matrices falls below this threshold. Table 1 demonstrates the **trade-off between accuracy and runtime** across varying tolerance thresholds. As the tolerance value decreases from 10^{-2} to 10^{-6}, the number of iterations and total runtime increase significantly. On average, both metrics increase by approximately six times. This highlights a **key bottleneck**: lower tolerance improves accuracy but also results in significantly higher computational cost. For example, on the Twitter graph, runtime increases from 11.7K seconds to over 60K seconds as tolerance tightens. These trade-offs motivate us to ask the following fundamental questions:

- *Can we design a non-iterative solution that addresses the zero-similarity problem without compromising the fundamental definition of SimRank?*
- *How can we solve the problem in less time while maintaining exactness?*
- *Can we apply the solution to real-world applications?*

To address these challenges, we propose **PJsim**, a scalable solution for the zero-similarity problem that provides exact, closed-form SimRank computations. Our contributions in this work are summarized as follows:

(1) We present the first closed-form, non-iterative solution for eliminating the costly recursive iterations and convergence checks inherent in existing SimRank methods. We reformulate the iterative SimRank equation as a Sylvester equation and provide efficient optimization methods.
(2) We investigate three techniques for solving the PJsim Sylvester equation: (i) **Neumann Series Approximation**, which trades off computation for accuracy by providing tunable parameters; (ii) **Eigenvalue Decomposition (EVD)**, which reduces linear algebraic work for diagonalizable matrices; and (iii) **Bartels-Stewart (BS) based block transformation**, which is unconstrained and achieves computational efficiency close to EVD.
(3) All three PJsim algorithms run in $O(n^3)$ complexity, which is significantly better than the $O(n^4)$ complexity of S_{Jeh} and the $O(Kn^3)$ complexity of S_{Yu}. We achieve up to **3.22× speedup** on the EVD approach and **2.15× speedup** on the BS approach. We also achieve higher semantic fidelity on both synthetic and real-world graphs, particularly in cases of zero-similarity. Our source code is available at https://github.com/SPADE-IITJ/PJsim.

2 Background

2.1 Existing Works

Previous attempts to address the zero-similarity problem have achieved only partial success, failing to provide comprehensive solutions to all manifestations of this fundamental limitation. A variety of algorithms have been proposed over the years, each addressing different aspects of similarity computation, trade-offs between accuracy and efficiency, and the zero-similarity problem. Table 2 summarizes these methods, highlighting their capabilities for all-pairs or single-source queries, time complexity, and effectiveness in solving the zero-similarity problem.

Zhao et al. [45] proposed **P-Rank** by incorporating both in- and out-links into the similarity computation, which reduces the number of node pairs with counter-intuitive zero similarities by considering bidirectional relationships. However, P-Rank still yields zero similarity when neither equidistant in-link nor out-link paths exist between nodes, demonstrating that simply expanding the path consideration is *insufficient to fully resolve the zero-similarity problem*. Chen and Giles [8] developed **ASCOS++** specifically to address the issue where *odd-length paths make no contribution* to SimRank scores, which represents a special case of the broader zero-similarity problem. While ASCOS++ provides a *sufficient condition for* $s(a, b) = 0$, it overlooks other critical conditions, particularly that even-length in-linked paths whose source nodes are not centered also lead to zero similarity, thus only partially resolving the fundamental issue.

Other link-based similarity measures, including **Random Walk with Restart (RWR)** [36, 37] and its variants such as Personalized PageRank [31, 44], also suffer from *SimRank-like zero-similarity issues*. As shown in Theorem 3 of [42], RWR assigns zero-similarity to a node pair (a, b) whenever no unidirectional path exists from b to a. This directional path requirement differs from S_{Jeh}'s parity-based behavior, where zero-similarity arises specifically from odd-length path connections. Despite these mechanistic differences, both limitations indicate that the zero-similarity problem extends beyond SimRank to affect a broader class of graph-based similarity measures.

Computational optimization research has addressed SimRank's high computational cost through several paradigms, yet none eliminate the fundamental trade-off between accuracy and efficiency in all-pairs computation. Lizorkin et al. [26] proposed three key optimizations: *essential node-pair selection*, *partial sums memoization*, and *threshold-sieved similarities*; reducing complexity from the

Table 2. Comparison of SimRank and related similarity algorithms addressing the zero-similarity problem. Here, $n = |V|$, $m = |E|$, K is the number of iterations, d' is the average in-degree after pruning, $\tilde{m}$ is the number of edges in the pruned graph, and r is the rank used in low-rank approximations. For iterative methods, time complexity is reported separately for sparse ($m = O(n)$) and dense ($m = O(n^2)$) graphs.

Algorithm	All-Pairs / Query	Time Complexity (Worst Case)	Solving Zero-Similarity	Approximate / Exact
P-Rank [45]	All-Pairs	$O(Kn^3)$	Partial	Approximate
ASCOS++ [8]	All-Pairs	$O(Kn^3)$	Partial	Approximate
RWR [36]	All-Pairs / Query	$O(Kn^3)$	No	Approximate
Lizorkin et al. [26]	All-Pairs	$O(Kn^3)$	No	Approximate
Yu et al. [41]	All-Pairs	$O(Kd'n^2)$	Partial	Approximate
Li et al. [25]	All-Pairs	$O(r^4n^2)$	No	Exact
TopSim [22]	Query	$O(K^2\tilde{m})$	No	Approximate
SimMat [11]	Query	$O(K^2\tilde{m})$	No	Exact
TSF [33]	Query	$O(K\tilde{m})$	No	Approximate
READS [20]	Query	$O(K\tilde{m})$	No	Approximate
SimRank* [42]	All-Pairs / Query	$O(Kn^3)$	Yes (Iterative)	Approximate
PJsim	All-Pairs / Query	$O(n^3)$	Yes (Exact, Non-Iterative)	Exact

prohibitive $O(Kd^2n^2)$ to $O(Knm)$ by eliminating redundant computations and exploiting graph structure. Building on this line of work, Yu et al. [41] further reduced the cost to $O(Kd'n^2)$ using *sophisticated topological sorting techniques* that enable fine-grained sharing of partial sums across iterations. In contrast, Li et al. [25] explored an *SVD-based matrix decomposition* approach with $O(r^4n^2)$ complexity, which degrades rapidly when high accuracy demands large rank values. Even **SimRank*** [42], which resolves the zero-similarity issue via *asymmetric influence propagation*, remains constrained by **iterative computation** and **convergence-dependent stopping criteria** in the all-pairs setting.

2.2 Preliminaries

2.2.1 ***SimRank* (S_{Yu}) Overview.*** The foundational SimRank measure, introduced by Jeh and Widom, quantifies the similarity between two nodes based on the similarity of their incoming neighbors. Formally, the similarity $s(a, b)$ between nodes a and b is defined as:

$$s(a,b) = \begin{cases} 1, & a = b, \\ c \cdot \dfrac{1}{|\mathcal{I}(a)|\,|\mathcal{I}(b)|} \displaystyle\sum_{(i,j)\in \mathcal{I}(a)\times\mathcal{I}(b)} s(i,j), & a \neq b, \end{cases}$$

In matrix form, this can be expressed as:

$$S_{Jeh} = \max\{c\,(Q\,S_{Jeh}\,Q^T),\ I_n\}$$

where S_{Jeh} is the SimRank similarity matrix, and Q is the backward transition matrix with entries:

$$Q_{ij} = \begin{cases} \frac{1}{|\mathcal{I}(i)|}, & \text{if } \exists \text{ edge } (j \to i) \in E; \\ 0, & \text{otherwise} \end{cases}$$

Li et al.'s SimRank model offers a variation with the recursive matrix form:

$$S_{Li} = c\,(Q\,S_{Li}\,Q^T) + (1-c)\,I_n$$

To overcome the "zero similarity" problem encountered in these earlier SimRank variations, Yu et al. proposed Geometric SimRank*, defined by the recursive matrix equation:

$$S_{\text{Yu}} = \frac{c}{2}\left(QS_{\text{Yu}} + S_{\text{Yu}}Q^T\right) + (1-c)\, I_n \tag{1}$$

The standard iterative approach to solve Equation 1 initializes

$$S_{\text{Yu}}^{(0)} = (1-c)\, I_n,$$

and iteratively updates

$$S_{\text{Yu}}^{(k)} = \frac{c}{2}\left(QS_{\text{Yu}}^{(k-1)} + S_{\text{Yu}}^{(k-1)}Q^T\right) + (1-c)\, I_n.$$

Convergence is achieved as k increases, yielding the similarity matrix S_{Yu}. However, the inherent recursive structure motivates the search for a direct, non-iterative solution.

2.2.2 ***Vectorization Operator (*** Vec ***).*** The vectorization operator [18], denoted as $Vec(A)$, transforms a matrix A of size $n \times n$ into a column vector of size $n^2 \times 1$ by stacking the columns of the matrix from top to bottom.

2.2.3 ***Kronecker Product (*** $\otimes$ ***).*** The Kronecker product of two matrices A (of size $m \times p$) and B (of size $q \times r$), denoted by $A \otimes B$, is a block matrix of size $mq \times pr$ formed by taking each element of A and multiplying it by the entire matrix B:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1p}B \\ a_{21}B & a_{22}B & \cdots & a_{2p}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mp}B \end{bmatrix}$$

The Kronecker product is a fundamental tool for linearizing bilinear matrix equations.

2.2.4 ***Kronecker Sum (*** $\oplus$ ***).*** The Kronecker sum of two square matrices A (of size $n \times n$) and B (of size $n \times n$) is defined as:

$$A \oplus B = (A \otimes I_n) + (I_n \otimes B)$$

The Kronecker sum arises naturally when vectorizing expressions involving both pre- and post-multiplication by matrices.

2.2.5 ***Neumann Series Inverse Approximation.*** The Neumann series [47] provides a method to approximate the inverse of a matrix through an infinite expansion. For any invertible matrix B, if the spectral radius satisfies $\rho(I - B) < 1$, then the inverse can be expressed as:

$$B^{-1} = \sum_{k=0}^{\infty} (I-B)^k = I + (I-B) + (I-B)^2 + (I-B)^3 + \cdots$$

where $\rho(\cdot)$ denotes the spectral radius (the largest absolute eigenvalue). The series converges geometrically when the convergence condition is satisfied. By truncating the series at K terms, we obtain an approximation $B^{-1} \approx \sum_{k=0}^{K} (I-B)^k$, where the accuracy improves as K increases. The Neumann series is particularly useful for avoiding explicit matrix inversion when dealing with large-scale systems, though convergence speed depends critically on how close $\rho(I - B)$ is to unity.

2.2.6 ***Lyapunov Formulation.*** To facilitate a closed-form solution, we apply the Vec operator and properties of the Kronecker product to transform Equation 1. The **Sylvester equation** [34] is a fundamental matrix equation of the form $AX + XB = C$, where $A \in \mathbb{R}^{m \times m}$, $B \in \mathbb{R}^{n \times n}$, $C \in \mathbb{R}^{m \times n}$, and $X \in \mathbb{R}^{m \times n}$ is the unknown matrix to be solved. The **Lyapunov equation** [12] is a special case of the Sylvester equation where $B = A^T$ and both matrices are square ($m = n$), yielding the form

$AX + XA^T = C$. This specialized structure arises naturally in stability analysis and control theory, and admits efficient solution methods that exploit its symmetry properties.

Applying the Vec operator to both sides of Equation 1 and using the property $\text{Vec}(AXB) = (B^T \otimes A)\text{Vec}(X)$, we can rewrite the equation. To directly relate it to the standard Lyapunov equation form, we first rearrange Equation 1 as:

$$\begin{aligned} & \tfrac{1}{2}(I_n S + S I_n) - \tfrac{c}{2}(Q S + S Q^T) && = (1-c) I_n \\ \Longrightarrow \quad & (I_n - c Q) S + S (I_n - c Q^T) && = 2 (1-c) I_n \\ \Longrightarrow \quad & X S + S X^T && = 2 (1-c) I_n \end{aligned} \tag{2}$$

where $X = I_n - c\,Q$. This is precisely a Lyapunov equation with $A = X$, $B = X^T$, and $C = 2(1-c)I_n$. We refer to Equation 2 as **PJsim**, and denote its solution as S_{PJ}. By considering the vectorized form of S_{PJ}, denoted as $s = \text{Vec}(S_{\text{PJ}})$, we can transform the PJsim equation into a linear system that can be solved directly. This transformation involves the Kronecker product and sum, as explored in the subsequent sections. The connection to the Lyapunov equation provides a theoretical foundation for finding an exact, non-iterative solution.

3 Methodology

We propose **PJsim**, a novel graph similarity measure that computes **exact closed-form node similarity scores**. Unlike existing iterative methods that are prone to convergence issues and approximation errors, PJsim ensures precision through a direct mathematical formulation. As introduced earlier in Equation 2, our approach is grounded in the continuous-time Lyapunov equation, which can be expressed as:

$$X\,S_{\text{PJ}} + S_{\text{PJ}}\,X^T = 2\,(1-c)\,I_n \tag{3}$$

where $X = (I_n - cQ)$ and S_{PJ} denotes the PJsim similarity matrix.

We begin by solving Equation 3 in a **direct, non-iterative closed form** and optimize it using three complementary techniques. First, the **Neumann Series Approximation** provides a *general, tunable solution* without structural assumptions, but may *converge slowly* when the spectral radius is close to one. To address this, we adopt an **Eigenvalue Decomposition (EVD)-based** method that **reduces algebraic operations** and improves runtime [30], at the cost of requiring diagonalizability and potential sensitivity to ill-conditioned spectra. Finally, we employ the **Bartels–Stewart block transformation**, which is *fast, exact, and applicable to all matrices*, achieving cubic time complexity without structural constraints. Each method has distinct trade-offs, but all deliver **exact solutions in $O(n^3)$ time**, outperforming existing iterative SimRank techniques.

3.1 PJsim: Direct Closed-Form Approach

We present our first approach to solve the PJsim's Lyapunov equation (Equation 3) by leveraging the properties of the Vec operator and the Kronecker product ($\otimes$). We begin by establishing the vectorized form of our PJsim.

Theorem 3.1 (**PJsim's Vectorized Lyapunov Equation**). *The PJsim's Equation 3 can be equivalently expressed in vectorized form as:*

$$(X \oplus X)\; Vec(S_{PJ}) = 2\,(1-c)\; Vec(I_n) \tag{4}$$

Proof. We apply the Vec operator to both sides of our PJsim Equation 3 and use the property that $\text{Vec}(ABC) = (C^T \otimes A)\text{Vec}(B)$. We have:

$$\text{Vec}(XS_{\text{PJ}} + S_{\text{PJ}}X^T) = \text{Vec}(2(1-c)I_n) \tag{5}$$

$$\text{Vec}(XS_{\text{PJ}}) + \text{Vec}(S_{\text{PJ}}X^T) = 2(1-c)\text{Vec}(I_n) \tag{6}$$

$$(I_n \otimes X)\text{Vec}(S_{\text{PJ}}) + (X \otimes I_n)\text{Vec}(S_{\text{PJ}}) = 2(1-c)\text{Vec}(I_n) \tag{7}$$

$$[(I_n \otimes X) + (X \otimes I_n)]\text{Vec}(S_{\text{PJ}}) = 2(1-c)\text{Vec}(I_n) \tag{8}$$

$$(X \oplus X)\text{Vec}(S_{\text{PJ}}) = 2(1-c)\text{Vec}(I_n) \tag{9}$$

where we used the definition of the Kronecker sum: $A \oplus B = (A \otimes I) + (I \otimes B)$. □

Corollary 3.1.1 (**PJsim Direct Closed-form Solution**). *The vectorized PJsim similarity matrix has the direct closed-form solution:*

$$Vec(S_{PJ}) = 2\,(1-c)\,(X \oplus X)^{-1}\,Vec(I_n) \tag{10}$$

We establish the validity of our PJsim formulation by proving that the required matrix inverse exists.

Theorem 3.2 (**PJsim Matrix Invertibility**). *For Equation 10 with $X = I_n - cQ$, where $c \in (0,1)$ and Q is a backward transition matrix, the matrix $X \oplus X$ is invertible.*

Proof. We analyze the eigenvalues of X and subsequently those of $X \oplus X$ for our PJsim method. The eigenvalues of $X = I_n - cQ$ lie in the range $[1-c, 1]$, since the eigenvalues of a backward transition matrix Q are bounded by $[0, 1]$, and we have $\lambda_i(X) = 1 - c\lambda_i(Q)$. The eigenvalues of the Kronecker sum $X \oplus X$ are given by $\{\lambda_i(X) + \lambda_j(X) : i, j = 1, \ldots, n\}$, which are the pairwise sums of eigenvalues of X. Therefore, the eigenvalues of $X \oplus X$ lie in the range $[2(1-c), 2]$. Since $c \in (0,1)$, we have $2(1-c) > 0$, ensuring that all eigenvalues of $X \oplus X$ are strictly positive. Hence, $X \oplus X$ is invertible, validating that the solution of Equation 10 exists. □

With Theorem 3.2, we establish that our PJsim closed-form solution in Equation 10 **yields exact similarity scores** without any *approximation*, which we have validated in Section 4.3.2. This represents a fundamental advantage of PJsim over iterative approaches that rely on convergence criteria and produce only approximate results. However, directly inverting the $n^2 \times n^2$ matrix $(X \oplus X)$ to compute $Vec(S_{\text{PJ}})$ has **prohibitive time complexity** of $O(n^6)$, rendering this direct approach impractical for large-scale graphs. To overcome this computational limitation while preserving the exactness of PJsim, we develop more efficient solution methods.

Remark 1 (**Sum Property**). *A distinctive mathematical property of our PJsim similarity matrix S_{PJ} derived from Equation 3 and Equation 10 is that the sum of all its elements equals the number of nodes n in the graph: $\vec{\mathbf{1}}^T S_{PJ}\vec{\mathbf{1}} = n$. To see this, we multiply the PJsim Lyapunov equation $XS_{PJ} + S_{PJ}X^T = 2(1-c)I_n$ on the left by $\vec{\mathbf{1}}^T$ and on the right by $\vec{\mathbf{1}}$, yielding $\vec{\mathbf{1}}^T XS_{PJ}\vec{\mathbf{1}} + \vec{\mathbf{1}}^T S_{PJ}X^T\vec{\mathbf{1}} = 2(1-c)n$. Since $X = I_n - cQ$ and $Q^T\vec{\mathbf{1}} = \vec{\mathbf{1}}$ for a backward transition matrix, we have $X^T\vec{\mathbf{1}} = (1-cq)\vec{\mathbf{1}}$. Letting $s = \vec{\mathbf{1}}^T S_{PJ}\vec{\mathbf{1}}$ and simplifying, we obtain $s \cdot 2(1-c) = 2(1-c)n$, which gives us $s = n$. This property provides a useful validation check for PJsim implementations.*

3.2 PJsim: Neumann Series Approximation

From Theorem 3.2, we establish that computing S_{PJ} reduces to finding the inverse of $(X \oplus X)$, a matrix $n^2 \times n^2$. However, directly inverting is computationally expensive. One approach to reduce this computational barrier is the Neumann series approximation [47], a technique commonly employed

for approximating matrix inverses. Instead of directly computing the inverse, the Neumann series can be used to approximate the inverse of $(X \oplus X)$ as

$$(X \oplus X)^{-1} = \sum_{k=0}^{\infty} \left[I_{n^2} - (X \oplus X)\right]^k,$$

provided the spectral radius satisfies $\rho(I_{n^2} - X \oplus X) < 1$. Applying this principle to our PJsim formulation from Corollary 3.1.1, we obtain an iterative approximation scheme that avoids explicit inversion of the large Kronecker sum matrix.

Theorem 3.3 (**PJsim Neumann Series Approximation**). *The PJsim similarity matrix from Equation 10 can be approximated using the Neumann series as:*

$$Vec(S_{PJ}) = 2(1-c) \sum_{k=0}^{\infty} A^k Vec(I_n), \tag{11}$$

where $A = -I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)$. In practice, we truncate at K terms:

$$Vec(S_{PJ}^{(K)}) = 2(1-c) \sum_{k=0}^{K} A^k Vec(I_n). \tag{12}$$

Proof. Starting from

$$(X \oplus X)^{-1} = \sum_{k=0}^{\infty} \left[I_{n^2} - (X \oplus X)\right]^k$$

We expand the Kronecker sum:

$$\begin{aligned} X \oplus X &= (I_n - cQ) \otimes I_n + I_n \otimes (I_n - cQ) \\ &= 2I_{n^2} - c(Q \otimes I_n + I_n \otimes Q). \end{aligned} \tag{13}$$

Therefore,

$$I_{n^2} - (X \oplus X) = -I_{n^2} + c(Q \otimes I_n + I_n \otimes Q). \tag{14}$$

Let $A = [-I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)]$, substituting into Equation 10 gives Equation (11), and truncating at K terms yields Equation (12). □

Convergence to Exact Solution. To obtain results as close to exact as possible, one must rely on the infinite series in Equation (11) rather than truncating at a fixed K. In practice, this means iterating until convergence is achieved, i.e., continuing iterations until $\|S_{\text{PJ}}^{(k)} - S_{\text{PJ}}^{(k-1)}\|_F < \epsilon$ for some tolerance ϵ. This approach minimizes the error between the Neumann approximation output and the exact closed-form solution, though the number of iterations required depends critically on the spectral properties of the iteration matrix.

The Neumann series converges if and only if

$$\rho\left(-I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)\right) < 1.$$

The eigenvalues of this iteration matrix are

$$\lambda_{ij} = -1 + c(\lambda_i(Q) + \lambda_j(Q)), \quad i, j = 1, \ldots, n. \tag{15}$$

For column-stochastic Q with $\lambda_{\max}(Q) = 1$, the maximum eigenvalue is

$$\lambda_{\max}\left(-I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)\right) = -1 + 2c. \tag{16}$$

For convergence, we require $|-1 + 2c| < 1$, which gives $0 < c < 1$. This condition is always satisfied for valid decay factors. However, the convergence *rate* depends critically on how close $\lambda_{\max}$ is to unity.

The intuition behind this convergence behavior is straightforward. When $c = 0.5$, the **dominant eigenvalue equals zero**, providing the **fastest convergence**. As c increases beyond 0.5, the dominant eigenvalue $-1 + 2c$ grows toward 1, causing the **spectral radius to approach unity**. For typical SimRank decay factors $c \in [0.6, 0.9]$, we have $\lambda_{\max} \in [0.2, 0.8]$, resulting in spectral radii that lead to extremely slow convergence. The number of iterations required grows approximately as $K \approx \frac{\log 1/\epsilon}{\log 1/\rho}$, where ϵ is the desired tolerance and ρ is the spectral radius. For $c = 0.8$, this can require **a higher number of iterations** for even modest accuracy, transforming what appears to be an elegant theoretical solution into a computationally prohibitive approach.

Based on Theorem 3.3, we summarize our Neumann series approximation for PJsim, referred to as $\text{PJsim}_{\text{NSA}}$, in Algorithm 1. The algorithm directly computes the truncated Neumann series in vector form. It iteratively applies the iteration matrix $A = -I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)$ to the previous term in the series and accumulates all terms to obtain the final approximation of $\text{Vec}(S_{\text{PJ}}^{(K)})$. The procedure begins by initializing $v^{(0)} = \text{Vec}(I_n)$, corresponding to the $k = 0$ term in the series, and sets this as the initial sum. At each iteration k from 1 to K, the algorithm computes $v^{(k)} = A\, v^{(k-1)}$ by exploiting the Kronecker product structure to avoid explicitly forming A, and adds it to the cumulative sum. After completing all K iterations, the sum is scaled by $2(1 - c)$ to produce the approximate PJsim similarity vector.

Algorithm 1 $\text{PJsim}_{\text{NSA}}$: Direct Truncated Neumann Solver

Require: c, Q, K (number of iterations)
Ensure: $\text{Vec}(S_{\text{PJ}}^{(K)})$

$I_n \leftarrow$ identity matrix of size n
$v^{(0)} \leftarrow \text{Vec}(I_n)$
Sum $\leftarrow v^{(0)}$
for $k = 1$ to K **do**
    Reshape $v^{(k-1)}$ into $n \times n$ matrix $V^{(k-1)}$
    Compute $U_1 \leftarrow QV^{(k-1)}$ and $U_2 \leftarrow V^{(k-1)}Q^T$
    $v^{(k)} \leftarrow -v^{(k-1)} + c(\text{Vec}(U_1) + \text{Vec}(U_2))$
    Sum $\leftarrow$ Sum $+ v^{(k)}$
$\text{Vec}(S_{\text{PJ}}^{(K)}) \leftarrow 2(1 - c)$ Sum
return $\text{Vec}(S_{\text{PJ}}^{(K)})$

3.2.1 ***Error Analysis.*** Truncating the infinite series at K terms introduces approximation error. Let $S_{\text{PJ}}^{(K)}$ denote the K-term truncation. The exact solution can be written as $S_{\text{PJ}} = S_{\text{PJ}}^{(K)} + \sum_{j=K+1}^{\infty} \Delta_j$, where Δ_j is the j-th term contribution. For a convergent series with spectral radius $\rho = \rho(A) < 1$, the tail sum satisfies $\| \sum_{j=K+1}^{\infty} \Delta_j \|_F \leq \frac{\|\Delta_{K+1}\|_F}{1-\rho}$. Since $\|\Delta_{K+1}\|_F \approx \rho \|\Delta_K\|_F$, the truncation error can be bounded by:

$$\|S_{\text{PJ}} - S_{\text{PJ}}^{(K)}\|_F \leq \frac{\rho^{K+1} \|S_{\text{PJ}}^{(0)}\|_F}{1 - \rho}. \tag{17}$$

This error bound reveals a critical limitation: when ρ approaches unity, the denominator $(1 - \rho)$ becomes very small, causing the **error bound to explode**. The alternating signs in the expansion due to the $-I_{n^2}$ term cause catastrophic cancellation where significant digits are lost in subtractive operations, further degrading accuracy beyond what the truncation error alone would suggest.

3.2.2 ***Complexity Analysis.*** Computing $\sum_{k=0}^{K} A^k \text{Vec}(I_n)$ can be done iteratively without explicitly forming powers of A. Starting with $v_0 = \text{Vec}(I_n)$, we compute $v_{k+1} = Av_k$ for $k = 0, 1, \ldots, K-1$. Each matrix-vector multiplication exploits the structure $A = -I_{n^2} + c(Q \otimes I_n + I_n \otimes Q)$ using the identities $(Q \otimes I_n)\text{vec}(V) = \text{vec}(QV)$ and $(I_n \otimes Q)\text{vec}(V) = \text{vec}(VQ^T)$. This requires reshaping the vector into an $n \times n$ matrix and performing two matrix-matrix multiplications, costing $O(n^3)$ per iteration. Computing K terms thus requires $O(Kn^3)$ time and $O(n^2)$ memory. However, the large K required for convergence (often hundreds to thousands of iterations for typical decay factors) makes even this optimized approach impractical.

3.2.3 ***Practical Limitations.*** Despite theoretical convergence for all valid decay factors $c \in (0, 1)$, the Neumann series approximation suffers from fundamental limitations. Convergence requires the necessary and sufficient condition $\rho(A) < 1$. Our experimental results in Section 4.3.1 show that none of the tested real graphs of Table 3 satisfy this condition, with the spectral radius consistently exceeding unity. This violation is structural and originates from the spectral properties of the transition matrix Q. Real-world graphs typically contain directed cycles, strongly connected components, or near-bipartite substructures, which induce eigenvalues with negative or complex real parts. Since the Kronecker-sum spectrum satisfies $\lambda_{ij}(A) = -1 + c(\lambda_i(Q) + \lambda_j(Q))$, the presence of such eigenvalues inevitably produces modes whose magnitude is at least one, thereby breaking the global contraction requirement of the Neumann series. Our objective was to compute **exact** PJsim results efficiently. However, $\text{PJsim}_{\text{NSA}}$ can only provide results that are **close to exact**; it remains an approximation method. The convergence rate deteriorates dramatically for typical SimRank parameters $c \in [0.6, 0.9]$, requiring hundreds to thousands of iterations with time complexity $O(Kn^3)$. This complexity is comparable to that of existing iterative SimRank methods, such as S_{Yu} (which also has $O(Kn^3)$ complexity), yet $\text{PJsim}_{\text{NSA}}$ still fails to deliver exact results. Numerical instability from alternating signs causes catastrophic cancellation, while the approximation error becomes uncontrollable when $\rho \to 1$ due to the exploding denominator $(1 - \rho)$ in the error bound. *The Neumann approximation offers no practical advantage, sacrificing exactness to deliver only approximate results with similar computational costs as existing methods, while being slow to converge, numerically unstable, and prone to uncontrolled errors.* These fundamental limitations motivate the development of more robust solution methods that can provide **exact results with predictable computational complexity**.

3.3 PJsim: Closed Form Solution via Eigenvalue Decomposition

Having established that the Neumann series approximation is still an iterative solution and does not provide exact solutions for PJsim, we now turn to exact solution methods. One natural approach to solving Equation 10 is through **Eigenvalue Decomposition (EVD)**, which exploits the spectral properties of the Kronecker sum. We now present our main theoretical result, which enables the fast and exact computation of the PJsim similarity matrix.

Theorem 3.4 (**PJsim's EVD-Based Solution**). *Assuming X is diagonalizable with eigenvalue decomposition $X = PDP^{-1}$, the PJsim similarity matrix can be expressed as:*

$$S_{PJ} = 2(1 - c)PZP^T \tag{18}$$

where Z is defined by $Vec(Z) = \mathbf{d} \odot Vec(P^{-1}(P^{-1})^T)$, and $\mathbf{d} = diag((D \oplus D)^{-1})$.

Proof. We start from the vectorized form in Corollary 3.1.1:

$$\text{Vec}(S_{\text{PJ}}) = 2(1 - c)(X \oplus X)^{-1}\text{Vec}(I_n) \tag{19}$$

Since $X = PDP^{-1}$, we have:

$$X \oplus X = (PDP^{-1}) \oplus (PDP^{-1}) = (P \otimes P)(D \oplus D)(P^{-1} \otimes P^{-1}) \tag{20}$$

Therefore:

$$(X \oplus X)^{-1} = (P \otimes P)(D \oplus D)^{-1}(P^{-1} \otimes P^{-1}) \tag{21}$$

Substituting this into our expression:

$$\mathrm{Vec}(S_{\mathrm{PJ}}) = 2(1-c)(P \otimes P)(D \oplus D)^{-1}(P^{-1} \otimes P^{-1})\mathrm{Vec}(I_n) \tag{22}$$

We let $\mathbf{d} = \mathrm{diag}((D \oplus D)^{-1})$ and note that $(P^{-1} \otimes P^{-1})\mathrm{Vec}(I_n) = \mathrm{Vec}(P^{-1}(P^{-1})^T)$.
Then:

$$\mathrm{Vec}(S_{\mathrm{PJ}}) = 2(1-c)(P \otimes P)[\mathbf{d} \odot \mathrm{Vec}(P^{-1}(P^{-1})^T)] \tag{23}$$

We define Z such that $\mathrm{Vec}(Z) = \mathbf{d} \odot \mathrm{Vec}(P^{-1}(P^{-1})^T)$, and use the property that $(A \otimes B)\mathrm{Vec}(C) = \mathrm{Vec}(ACB^T)$:

$$\mathrm{Vec}(S_{\mathrm{PJ}}) = 2(1-c)\mathrm{Vec}(PZP^T) \tag{24}$$

Therefore, $S_{\mathrm{PJ}} = 2(1-c)PZP^T$. □

Based on Theorem 3.4, we summarize our EVD-based solution for PJsim, referred to as PJsim$_{EVD}$, in Algorithm 2. We begin by computing the eigenvalue decomposition of matrix X (Line 1), followed by constructing the Kronecker sum matrix K (Line 2). Next, we extract the diagonal of K^{-1} and perform an element-wise Hadamard product with the vectorized form of M (Lines 3-4). The resulting vector is reshaped into a matrix Z (Line 6), which is finally used to compute S_{PJ} (Line 7).

Algorithm 2 PJsim$_{EVD}$: PJsim via Eigenvalue Decomposition

Require: X
Ensure: $S_{\mathrm{PJ}} = 2(1-c) \cdot PZP^T$
1: $X \leftarrow PDP^{-1}$ ▷ Eigenvalue decomposition
2: $K \leftarrow D \oplus D$
3: $\mathbf{d} \leftarrow \mathrm{diag}(K^{-1})$
4: $M \leftarrow P^{-1}(P^{-1})^T$
5: $\mathrm{vec}(Z) \leftarrow \mathbf{d} \odot \mathrm{vec}(M)$ ▷ $\odot$ is Element-wise product
6: Reshape $\mathrm{vec}(Z)$ into $Z \in \mathbb{R}^{n \times n}$
7: $S_{\mathrm{PJ}} \leftarrow 2(1-c) \cdot PZP^T$
8: **return** S_{PJ}

3.3.1 Complexity Analysis. We analyze each step of PJsim$_{EVD}$ (Algorithm 2) for PJsim computation. The eigenvalue decomposition of the $n \times n$ matrix X requires $O(n^3)$ operations. Computing $D \oplus D$ takes $O(n^2)$ time, while the element-wise inverse of the diagonal requires $O(n^2)$ operations. The computation of $P^{-1}(P^{-1})^T$ involves matrix multiplication with complexity $O(n^3)$. The element-wise product operation takes $O(n^2)$ time, and reshaping the vectorized form back to matrix form requires $O(n^2)$ operations. Finally, the matrix multiplication PZP^T contributes $O(n^3)$ complexity. The overall time complexity is dominated by the eigenvalue decomposition and matrix multiplications, each contributing $O(n^3)$, resulting in $O(n^3)$ total complexity. For space complexity, we need to store the eigenvectors P requiring $O(n^2)$ space, eigenvalues D requiring $O(n)$ space, and intermediate matrices M, Z, and S_{PJ} each requiring $O(n^2)$ space. Therefore, the space complexity is $O(n^2)$.

This $O(n^3)$ complexity represents a significant improvement over the $O(n^6)$ complexity of direct inversion of $(X \oplus X)$, thus justifying the PJsim EVD-based approach when X is diagonalizable. Matrices are guaranteed to be diagonalizable if they are real-symmetric, Hermitian, or more generally, normal (satisfying $XX^T = X^TX$). However, the backward-transition-derived matrix X in PJsim is typically non-normal because it represents directed graph structure with inherently asymmetric edge patterns. Determining whether X is diagonalizable requires computing its eigen

value decomposition, which itself incurs $O(n^3)$ cost. Consequently, the total runtime of $\text{PJsim}_{\text{EVD}}$ includes both this diagonalizability check and the subsequent execution of Algorithm 2. *The primary limitation of the PJsim EVD-based approach remains its reliance on the diagonalizability of the matrix X.* For cases where X is not diagonalizable, this method is not applicable, necessitating the exploration of alternative techniques that do not depend on any special cases of the graph.

3.4 PJsim: Closed Form Solution via Bartels-Stewart Algorithm

Returning to Equation 3, we recognize its resemblance to the **continuous-time Lyapunov equation** $AS + SA^T = C$, a specific instance of the **Sylvester equation** $AS + SB = C$. This class of equations is extensively studied in control systems and stability analysis [5, 12, 34]. Among the prominent algorithms for solving Lyapunov equations are the **Bartels-Stewart algorithm** [4] and **Hammarling's algorithm** [16]. However, Hammarling's algorithm imposes restrictions on the matrix A (stability) and C (negative semi-definiteness), which may not hold for our PJsim problem, making the Bartels-Stewart algorithm a more suitable choice.

As we noted in Section 3.1, a direct solution of the vectorized form of Equation 3 via matrix inversion has a prohibitive time complexity of $O(n^6)$. The Bartels-Stewart algorithm circumvents this by employing the **real Schur decomposition** [1, 9, 21, 27] to transform the matrix X into a **quasi-upper triangular form**, which allows for a more efficient solution of the resulting linear system through backward substitution.

Theorem 3.5 (**PJsim Schur-Based Lyapunov Transformation**). *Let X have real Schur decomposition $X = VUV^T$ where U is quasi-upper triangular and V is orthogonal. Then the PJsim's Equation 3 is equivalent to the transformed equation:*

$$UY + YU^T = 2(1-c)I_n \tag{25}$$

where $Y = V^T S_{PJ} V$ and the solution is recovered by $S_{PJ} = VYV^T$.

Proof. We start with the real Schur decomposition of the matrix X:

$$X = VUV^T \tag{26}$$

where U is a real quasi-upper triangular matrix with diagonal blocks of size 1×1 or 2×2, and V is a real orthogonal matrix ($V^T V = I_n$). Substituting this into Equation 3, we premultiply by V^T and postmultiply by V, and let $Y = V^T S_{\text{PJ}} V$:

$$V^T(VUV^T S_{\text{PJ}})V + V^T(S_{\text{PJ}} VU^T V^T)V = V^T W V \tag{27}$$

$$U(V^T S_{\text{PJ}} V) + (V^T S_{\text{PJ}} V)U^T = V^T(2(1-c)I_n)V \tag{28}$$

$$UY + YU^T = 2(1-c)V^T V \tag{29}$$

$$UY + YU^T = 2(1-c)I_n \tag{30}$$

Since V is orthogonal, we have $V^T V = I_n$, and S_{PJ} can be recovered by $S_{\text{PJ}} = VYV^T$. □

Theorem 3.6 (**PJsim Block Structure Properties**). *When $W = 2(1-c)I_n$ and U is quasi-upper triangular with block structure corresponding to eigenvalue blocks, the transformed PJsim equation $UY + YU^T = W$ admits a block-wise solution where diagonal blocks can be computed independently and off-diagonal blocks follow from backward substitution.*

Proof. We consider the block partitioning of W, Y, and U based on the quasi-upper triangular structure of U. Since $W = 2(1-c)I_n$, we have W as a block diagonal matrix where $W_{ii} = 2(1-c)I_{b_i}$ and $W_{ij} = 0$ for $i \neq j$, where b_i is the size of the i-th diagonal block.

For the general 2×2 block structure, we can write:

$$W = \begin{bmatrix} W_{bb} & 0 \\ 0 & W_{ss} \end{bmatrix}, \quad Y = \begin{bmatrix} Y_{bb} & Y_{bs} \\ Y_{sb} & Y_{ss} \end{bmatrix}, \quad U = \begin{bmatrix} U_{bb} & U_{bs} \\ 0 & U_{ss} \end{bmatrix}$$

The equation $UY + YU^T = W$ expands to the system:

$$W_{bb} = U_{bb}Y_{bb} + U_{bs}Y_{sb} + Y_{bb}U_{bb}^T + Y_{bs}U_{bs}^T \tag{31}$$

$$0 = U_{bb}Y_{bs} + U_{bs}Y_{ss} + Y_{bs}U_{ss}^T \tag{32}$$

$$0 = U_{ss}Y_{sb} + Y_{sb}U_{bb}^T + Y_{ss}U_{bs}^T \tag{33}$$

$$W_{ss} = U_{ss}Y_{ss} + Y_{ss}U_{ss}^T \tag{34}$$

We observe that the **last equation is a smaller Lyapunov equation** that can be solved independently. Once Y_{ss} is known, the second and third equations become linear systems that can be solved for Y_{bs} and Y_{sb}. Finally, the first equation reduces to a smaller Lyapunov equation. This **recursive structure enables backward substitution**. □

The Bartels-Stewart algorithm, adapted for our PJsim problem, leverages these theoretical results to provide an efficient computational solution. We can visualize the process as a repetition of computing four linear matrix equations of decreasing sizes, as demonstrated in the proof above.

Theorem 3.7 (**PJsim Small Block Lyapunov Solvability**). *For diagonal blocks U_{ii} of size $b_i \in \{1, 2\}$ arising from the quasi-upper triangular decomposition, equation $U_{ii}Y_{ii} + Y_{ii}U_{ii}^T = W_{ii}$ can be solved exactly using the vectorized form in less complexity.*

Proof. For blocks of size 1×1, the equation reduces to a scalar equation $2u_{ii}y_{ii} = w_{ii}$, giving $y_{ii} = \frac{w_{ii}}{2u_{ii}}$.

For blocks of size 2×2, we can apply the vectorized solution:

$$\text{Vec}(Y_{ii}) = (I_2 \otimes U_{ii} + U_{ii}^T \otimes I_2)^{-1}\text{Vec}(W_{ii})$$

This involves inverting a 4×4 matrix, which requires only $O(1)$ operations and is computationally trivial compared to the original $O(n^6)$ complexity. □

We illustrate the dependency-driven order of block-wise computations in the matrix Y in Figure 2. After transforming the Lyapunov equation into the Schur basis, the computation proceeds sequentially over block indices. At iteration t, the diagonal block $Y_{ss}^{(t)}$ is computed first, after which the off-diagonal blocks $Y_{js}^{(t)}$ for $j < s$ are obtained via backward substitution. Each $Y_{js}^{(t)}$ depends on the current diagonal block $Y_{ss}^{(t)}$ as well as blocks $Y_{ks}^{(t-1)}$ computed in earlier iterations with $k > j$. The corresponding symmetric blocks are then obtained by enforcing symmetry, i.e., $Y_{sj}^{(t)} = (Y_{js}^{(t)})^T$. This process repeats from the bottom-right block toward the top-left until the entire matrix Y is solved. The strict data dependencies across iterations make PJsim$_{BS}$ inherently sequential, leaving the initial Schur decomposition as the primary opportunity for parallelization.

We outline the Bartels-Stewart-based computation of PJsim, referred to as PJsim$_{BS}$, in Algorithm 3. We first compute the Schur decomposition of matrix X (Line 1), and transform the right-hand side matrix W into the Schur basis (Line 2). The algorithm proceeds in a block-wise manner, starting from the last block and iterating in reverse (Line 4). For each block i, we isolate the diagonal block W_{ss} and solve for Y_{ss} using the inverse of the Kronecker sum $U_{ss} \oplus U_{ss}$ (Line 6). The off-diagonal blocks are handled recursively by updating intermediate matrices D_j (Lines 7-11), followed by solving smaller Sylvester equations. Symmetry is enforced explicitly (Line 12). The matrices are then compressed for the next iteration (Line 13). After the block-wise computation concludes, the final PJsim matrix is obtained by transforming Z back using the orthogonal matrix E (Line 15).

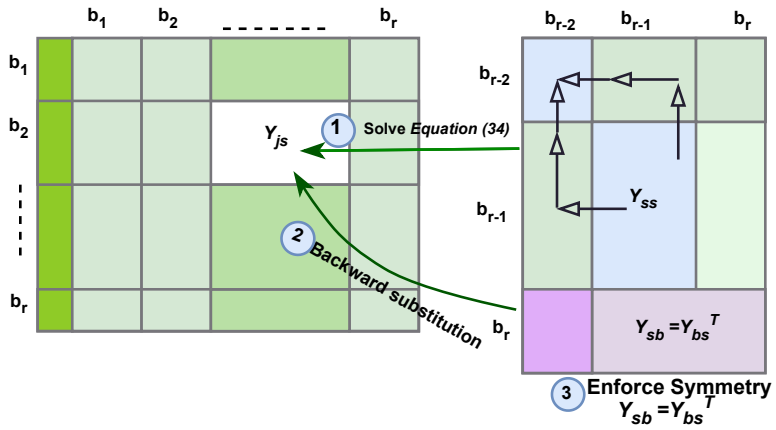


Fig. 2. Dependency-driven order of block-wise computation in the matrix Y after Schur transformation. Diagonal blocks are solved first, followed by off-diagonal column blocks via backward substitution, with symmetric blocks obtained by transposition.

Algorithm 3 PJsim$_{BS}$: PJsim via Bartels-Stewart Algorithm

Require: X
Ensure: $S_{\text{PJ}} \leftarrow EZE^T$

Compute Schur decomposition: $X = EUE^T$
Transform RHS: $W \leftarrow E^T WE$
Initialize $Z \leftarrow 0$, $Y \leftarrow Z$
for $i = r$ to 1 **do**
  Partition U, W, Y into blocks $\{b_1, \ldots, b_r\}$; extract U_{ss}, W_{ss}, Y_{ss}
  $Y_{ss} \leftarrow \text{reshape}((U_{ss} \oplus U_{ss})^{-1} \cdot \text{vec}(W_{ss}))$
  $D_{bs} \leftarrow W_{bs} - U_{bs} Y_{ss}$
  Partition D_{bs} into $\{D_j\}_{j=1}^{i-1}$
  for $j = i - 1$ to 1 **do**
    $D_j \leftarrow D_j - \sum_{k=j+1}^{i-1} U_{jk} Y_k$
    $Y_j \leftarrow \text{reshape}((U_{ss} \oplus U_{jj})^{-1} \cdot \text{vec}(D_j))$
  $Y_{21} \leftarrow Y_{12}^T$
  $W \leftarrow W_{bb} - U_{bs} Y_{sb} - Y_{bs} U_{bs}^T$
  $U \leftarrow U_{bb}$, $Y \leftarrow Y_{bb}$
$S_{\text{PJ}} \leftarrow EZE^T$
return S_{PJ}

3.4.1 Complexity Analysis. We analyze the major computational steps of PJsim$_{BS}$ (Algorithm 3) for PJsim computation. The **dominant step** is computing the **real Schur decomposition** of the $n \times n$ matrix X, which requires $O(n^3)$ operations. The **block-wise solution** involves **solving small Lyapunov equations** for diagonal blocks of size at most 2×2, which takes $O(1)$ per block. Since there are at most n such blocks, this contributes $O(n)$ complexity. The computation of **off-diagonal**

blocks through backward substitution involves matrix operations that collectively contribute $O(n^3)$ complexity. Finally, the **matrix multiplication** $S_{\text{PJ}} = EZE^T$ requires $O(n^3)$ operations. The overall time complexity is dominated by the Schur decomposition and final matrix multiplication, resulting in $O(n^3)$ total complexity.

For space complexity, we need to store the Schur decomposition matrices E and U, requiring $O(n^2)$ space each, intermediate matrices W, Y, and Z, requiring $O(n^2)$ space each, and the final PJsim similarity matrix S_{PJ}, requiring $O(n^2)$ space. The block partitioning operations (Lines 5-14 of Algorithm 3) reuse existing storage without allocating additional full-size matrices. Therefore, the space complexity is $O(n^2)$. Importantly, PJsim$_{BS}$ is more space-efficient than the iterative S_{Yu} formulation, which requires maintaining two complete similarity matrices at each iteration to compute convergence (the current similarity matrix $S^{(k)}$ and the previous similarity matrix $S^{(k-1)}$ to check $\|S^{(k)} - S^{(k-1)}\|_F < \epsilon$), resulting in $2n^2$ storage for similarity tracking alone, in addition to intermediate computation matrices. In contrast, PJsim$_{BS}$ computes the exact solution in a single pass without iteration, requiring only $O(n^2)$ storage with no redundant matrix copies.

Unlike the PJsim$_{EVD}$ algorithm, the PJsim$_{BS}$ algorithm does not require the diagonalizability of matrix X, making it universally applicable to all graph structures while maintaining $O(n^3)$ time complexity and $O(n^2)$ space complexity. This $O(n^3)$ complexity makes the PJsim$_{BS}$ algorithm a **practical and exact** approach for computing exact similarity scores, especially for non-diagonalizable transition matrices. We have successfully overcome the limitations of the PJsim$_{EVD}$ algorithm while maintaining the same computational complexity and achieving better space efficiency than iterative methods.

4 Implementation and Evaluation

In this section, we discuss our implementation strategies, followed by the experimental results.

4.1 Platform

We use a server with two AMD EPYC 7742 64-core processors (2.25 GHz, x86_64), providing 128 CPU cores (2 threads per core) across 2 sockets, with 512 MB L3 cache and 512 GB DDR4 memory, running Ubuntu 22.04. For fair comparison, we implement both S_{Yu} and S_{PJ} in C++, using STL containers like vectors for graph structures. We use the Eigen library for calculating Schur decomposition in S_{PJ}, and LAPACK 3.12.1 for fast matrix multiplications in both S_{PJ} and S_{Yu}.

4.2 Dataset

We evaluate our algorithm on both real and synthetic graphs, including R-MAT graphs [7]. The real graphs contain between 4.7K and 525.3K nodes, and between 39.9K and 3.02M edges. The synthetic R-MAT graphs span edge counts from 0.1M to 100M. All graphs are directed and unweighted. Table 3 summarizes their properties.

Since no structural criterion from graph topology guarantees the diagonalizability of X, we manually generated synthetic graphs (G13–G16) whose X matrices were constructed to be diagonalizable and thus are EVD-compatible. This selection covers a broad spectrum of graph densities. Graph density directly impacts SimRank computation: in denser graphs, a larger number of incoming neighbors increases the pairwise comparison space, leading to higher computation time, whereas in sparser graphs, the number of contributing paths is smaller, resulting in fewer non-zero similarity entries and faster computation.

4.3 Experimental Results

4.3.1 ***Runtime Performance of PJsim.*** As discussed in Section 3.2.3, the Neumann series approximation (NSA) requires $\rho(A) < 1$ for guaranteed convergence, a condition not satisfied by

Table 3. Graphs used in the experiments. $|V|$ and $|E|$ denote the number of nodes and edges, respectively, and $|E|/|V|$ is the average degree. Real graphs are from SNAP [24], and synthetic graphs are generated using R-MAT [7]. The suffixes K and M denote thousand and million, respectively.

Abbreviation	Graph	$\|V\|$	$\|E\|$	$\|E\|/\|V\|$
Real-world Graphs				
G1	Wikipedia	4.7K	185.4K	39.45
G2	Gnutella04	10.8K	39.9K	3.69
G3	DBLP	12.6K	49.8K	3.95
G4	Cit-HepPh	34.5K	420.9K	12.20
G5	Twitter	81.3K	1.76M	21.65
G6	Orkut	117.2K	1.08M	9.22
G7	Amazon	260.2K	2.35M	9.0
G8	Delaunay	525.3K	3.02M	5.7
R-MAT Graphs				
G9	0.1M_edges	50K	0.1M	2.00
G10	1M_edges	100K	1M	10.00
G11	10M_edges	500K	10M	20.00
G12	100M_edges	1M	100M	100.00
R-MAT Graphs (EVD-Compatible)				
G13	5M_edges	1K	0.1M	100.00
G14	20M_edges	5K	2M	400.00
G15	50M_edges	10K	50M	5000.00
G16	100M_edges	20K	100M	5000.00

the real graphs in our experiments. To verify correctness under its theoretical assumptions, we evaluated PJsim$_{NSA}$ on modified versions of graphs G1, G2, G9, and G10, where cycles were removed, and the transition matrix was adjusted to enforce $\rho(A) < 1$. As shown in Figure 3, PJsim$_{NSA}$ converges to the same fixed-point solution as S_{Yu}, confirming its numerical correctness. In practice, however, NSA typically requires more iterations to propagate similarity information across the full n^2-dimensional space, whereas S_{Yu} benefits from more localized iterative refinement, motivating the use of alternative closed-form solvers such as PJsim$_{BS}$ and PJsim$_{EVD}$.

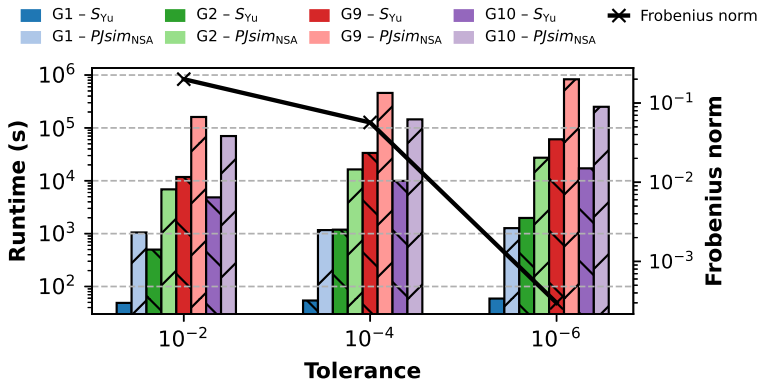


Fig. 3. Execution time (in seconds) on left Y-axis and average Frobenius-norm on right Y-axis of PJsim$_{\text{NSA}}$ and S_{Yu} across G1, G2, G9, and G10.

To evaluate the runtime performance of PJsim$_{\text{BS}}$ and PJsim$_{\text{EVD}}$, we computed both PJsim and S_{Yu} for a fixed decay factor of $c = 0.6$. Since the objective was to compare the total time required to obtain accurate similarity results, we ran S_{Yu} until convergence; S_{Yu} is iterated until the Frobenius-norm difference between successive similarity matrices becomes negligibly small and the SimRank

values become saturated over successive iterations, indicating that further iterations no longer change the values in any meaningful way. Across our test graphs, convergence required between 15 iterations (for sparse graphs) and up to 67 iterations (for denser graphs), with an average of approximately 35 iterations.

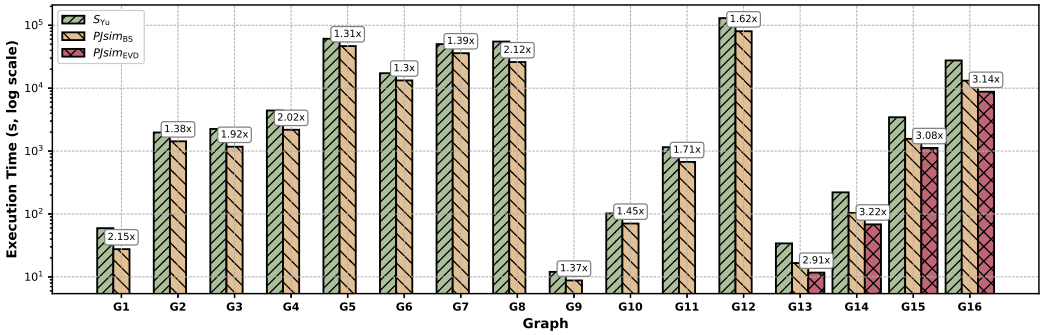


Fig. 4. Execution time (in seconds) of PJsim and S_{Yu} across all graphs. Both methods used a decay factor of $c = 0.6$. For EVD-compatible graphs (G13–G16), we additionally report the runtime of the eigenvalue-based variant, PJsim_{EVD}.

As shown in Figure 4, both the variants of PJsim consistently outperformed S_{Yu} across all graphs. Our analysis focuses on PJsim_{BS} and $\text{PJsim}_{\text{EVD}}$, as they provide exact solutions while maintaining fast runtimes. On real-world graphs, PJsim_{BS} achieved speedups ranging from 1.3× to 2.2×, while for synthetic graphs, the speedups were between 1.4× and 2.2×, depending on **graph sparsity and size**. On average, PJsim_{BS} was approximately 1.6× faster overall. The **maximum speedups** were observed on denser graphs, where PJsim_{BS}'s reliance on **closed-form operations** such as matrix multiplication and Schur decomposition proved highly effective. In contrast, on sparser graphs, although PJsim still achieves better runtime, the observed speedup is smaller. This is because sparsity reduces the effective workload per iteration in S_{Yu}. The transition matrix contains fewer nonzero interactions, thereby accelerating convergence by limiting the number of updates that contribute to similarity propagation. As a result, the iterative method runs faster on sparse graphs, narrowing the performance gap.

PJsim_{BS} achieves better runtime performance by **significantly reducing the number of floating-point operations**. This difference becomes clear when comparing Equation 1 (Yu's iterative formulation) and PJsim_{BS} (Algorithm 3). Yu's iterative formula involves **two matrix multiplications and one matrix addition per iteration**. If K iterations are required for convergence, the total cost becomes $2K$ matrix multiplications and K additions. Each matrix multiplication between two $n \times n$ matrices requires $O(n^3)$ operations, resulting in a cumulative cost of $O(Kn^3)$. When K is large, as in the case of denser graphs or tighter convergence thresholds, this cost becomes substantial. In contrast, **PJsim_{BS} avoids iterative updates**. PJsim_{BS} relies on a single Schur decomposition followed by a transformation and triangular solver. The Schur decomposition has a one-time cost of $O(n^3)$, and the subsequent back-substitution steps remain within the same order. Crucially, there are no repeated iterations, which bounds the total cost to a constant multiple of n^3. *This reduction in repeated computation makes PJsim more efficient in practice, especially on larger and denser graphs.*

Importantly, this behavior highlights the **scalability** of PJsim. As graph size increases, the relative speedup of PJsim_{BS} over S_{Yu} does not deteriorate; instead, it remains stable or increases

across both real-world and synthetic graphs. This trend indicates that the one-time $O(n^3)$ cost of the closed-form solution amortizes well as the problem size grows, whereas the cumulative cost of iterative refinement in S_{Yu} increases with both graph size and the number of iterations required for convergence. Consequently, PJsim's performance advantage becomes more pronounced on larger and denser graphs, demonstrating its practical scalability beyond small benchmark instances.

We also evaluated the eigenvalue-based variant $\text{PJsim}_{\text{EVD}}$, which assumes that the backward transition matrix X is diagonalizable (see Theorem 3.4). On EVD-compatible graphs, $\text{PJsim}_{\text{EVD}}$ **achieved the best performance among all variants**, consistently outperforming both PJsim_{BS} and S_{Yu}. Across these graphs, $\text{PJsim}_{\text{EVD}}$ achieved speedups ranging from 2.9× to 3.2× over S_{Yu}, with an average speedup of approximately 3.1×. This performance stems from its **minimal runtime footprint**: instead of using Schur decomposition, it **directly diagonalizes** X and computes the similarity matrix using closed-form algebraic operations. Because no linear systems are solved and no iterative refinement is required, $\text{PJsim}_{\text{EVD}}$ performs significantly fewer floating-point operations, making it the **fastest method overall** when its assumptions hold.

Table 4. Frobenius norm between PJsim variants (PJsim_{BS} and $\text{PJsim}_{\text{EVD}}$) and S_{Yu} under different tolerances. Lower values indicate higher similarity. MAE is reported at 10^{-6}.

	10^{-2}		10^{-4}		10^{-6}		MAE (10^{-6})	
Graph	PJsim_{BS}	$\text{PJsim}_{\text{EVD}}$	PJsim_{BS}	$\text{PJsim}_{\text{EVD}}$	PJsim_{BS}	$\text{PJsim}_{\text{EVD}}$	PJsim_{BS}	$\text{PJsim}_{\text{EVD}}$
G1	0.10	-	0.058	-	0.00012	-	0.000018	-
G2	0.12	-	0.060	-	0.00018	-	0.000021	-
G3	0.18	-	0.072	-	0.00031	-	0.000036	-
G4	0.14	-	0.068	-	0.00020	-	0.000024	-
G5	0.11	-	0.053	-	0.00007	-	0.000009	-
G6	0.15	-	0.065	-	0.00015	-	0.000019	-
G7	0.21	-	0.076	-	0.00042	-	0.000046	-
G8	0.23	-	0.078	-	0.00045	-	0.000051	-
G9	0.14	-	0.068	-	0.00034	-	0.000042	-
G10	0.24	-	0.080	-	0.00040	-	0.000048	-
G11	0.19	-	0.074	-	0.00038	-	0.000039	-
G12	0.27	-	0.085	-	0.00056	-	0.000057	-
G13	0.14	0.14	0.028	0.028	0.00071	0.00072	0.000012	0.000012
G14	0.24	0.23	0.016	0.016	0.00094	0.00094	0.000015	0.000015
G15	0.28	0.28	0.023	0.021	0.00143	0.00143	0.000021	0.000021
G16	0.30	0.31	0.037	0.037	0.00195	0.00195	0.000034	0.000034

4.3.2 ***Accuracy performance of PJsim.*** To assess the accuracy of PJsim, we compared its similarity matrix S_{PJ} against S_{Yu}, the result of the iterative SimRank* method. For each graph, we executed Yu et al.'s algorithm at three tolerance levels (10^{-2}, 10^{-4}, and 10^{-6}), while PJsim was computed once using its closed-form formulation. All experiments were run using double-precision arithmetic to avoid rounding errors. Table 4 reports the **Frobenius norm** between the two matrices under different convergence thresholds. Additionally, we compute the Mean Absolute Error (MAE) at the strictest tolerance (10^{-6}). The Frobenius norm between two matrices A and B is defined as, $\|A - B\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (A_{ij} - B_{ij})^2}$. This norm aggregates global matrix-wise error and is particularly suitable for comparing similarity matrices, where indirect structural effects are non-local and cumulative. Unlike pointwise metrics, the Frobenius norm captures the overall deviation in latent similarity propagation, providing a holistic sense of alignment between matrices.

We observe that as the tolerance in the iterative algorithm is tightened from 10^{-2} to 10^{-6}, the **Frobenius norm consistently decreases**. This indicates that the iterative solution gradually converges toward PJsim's result. At 10^{-6}, the **difference is almost negligible**, with norms falling below 0.005 in all cases and MAE values consistently under 0.0005. This validates that PJsim's output

matches the fully converged solution of S_{Yu} without requiring any iteration. It is important to note that PJsim operates without approximation or threshold tuning. It provides a deterministic, closed-form solution to the SimRank* formulation by solving the continuous-time Lyapunov equation using the Bartels-Stewart algorithm. Unlike iterative approaches, PJsim makes no assumptions about the graph structure and is free of issues such as early stopping or numerical drift.

Across both **sparse graphs (e.g., G2, G3)** and **denser topologies (e.g., G1, G5)**, PJsim maintains **strong agreement** with S_{Yu}. The slightly **higher norms in dense graphs** stem from the **iterative method requiring more steps to propagate indirect similarities accurately** due to increased path branching. PJsim, in contrast, directly accounts for all such interactions in a single matrix computation, making it a robust and reliable baseline. Thus, PJsim not only achieves high fidelity to converged similarity scores but also serves as a computational benchmark. Its matrix-level accuracy, consistency across graph types, and elimination of convergence dependencies highlight its utility for both evaluation and deployment.

On EVD-compatible graphs (G13 to G16), both PJsim_{BS} and $\text{PJsim}_{\text{EVD}}$ are executed. Even though both approaches solve the same Lyapunov equation using different decompositions, they produce nearly identical results. The Frobenius norms and MAE values for these graphs are effectively the same, confirming that **both versions of PJsim converge to the same similarity matrix**. Among all methods tested, $\text{PJsim}_{\text{EVD}}$ achieved the fastest runtime while maintaining the same level of accuracy as PJsim_{BS}, making it the preferred choice whenever eigenvalue decomposition is applicable.

4.4 Case Study

To validate the real-world applicability and semantic alignment of PJsim, we conducted two case studies focusing on different types of graph-based tasks.

4.4.1 ***Semantic Fidelity on Citation Graphs.*** To assess how well PJsim aligns **structural similarity** with semantic meaning, we applied it to three citation datasets: CitH (directed) [23, 24], DBLP_V1 (undirected, binary classification), and DBLP_V12 (undirected, multi-class) [39]. In this evaluation, we treat the externally provided ground-truth domain labels associated with each node as the reference meaning, and assess whether similarity rankings align with these real-world groupings. For each dataset, we randomly selected 2000 query graphs and computed pairwise similarity scores using PJsim, S_{Yu}, S_{Jeh}, and RWR. We evaluate **semantic fidelity** by measuring how closely the ranked list of similar nodes for a query aligns with its ground-truth label set, using Kendall's τ, Spearman's ρ, and NDCG@10. Additionally, **semantic consistency** is computed by taking each query node, retrieving its top-k neighbors ranked by similarity score, and measuring the fraction that shares the query's ground-truth label, averaged over all queries. Figure 5 presents the semantic fidelity results across the three datasets. Notably, PJsim and S_{Yu} produce nearly identical semantic results because both converge to the same SimRank solution, with PJsim achieving this more efficiently through its closed-form computation.

On **CitH, PJsim achieves scores of 0.312 (Kendall), 0.337 (Spearman), and 0.7921 (NDCG)**, showing a **moderate edge over competing methods**. However, all methods report relatively similar NDCG values on CitH, indicating that this dataset offers limited **semantic granularity**. When the label space is coarse rather than fine-grained, the evaluation becomes less discriminative, making it harder for structural methods to separate semantically meaningful graph pairs.

In contrast, **PJsim performs significantly better on DBLP_V12, where it achieves 0.3105 (Kendall), 0.3458 (Spearman), and 0.816 (NDCG)**, outperforming all baselines by a substantial margin. This improvement stems from the **richer semantic structure in DBLP_V12**, which includes papers from diverse sources such as DBLP, ACM, and Microsoft Academic Graph. The

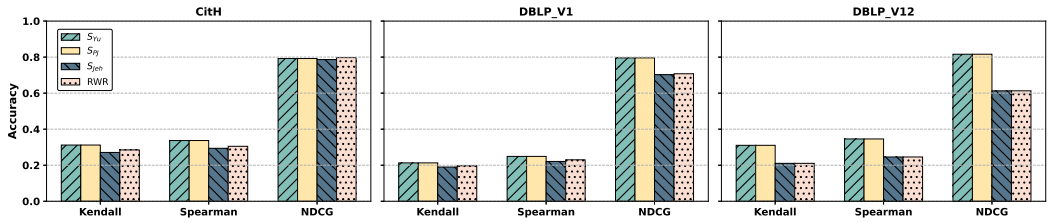


Fig. 5. Quantitative results of semantic effectiveness on citation graphs.

Table 5. Performance Comparison of Link Prediction Methods. Metrics include AUC (area under the curve), P@K (precision at top-K), AUPR (area under precision–recall), and Acc. (accuracy). Benchmarks are set up by GraphSAGE, marked as blue. PJsim is giving the best results and is closest to benchmarks, marked as green.

	Cosine				Jaccard				SimRank				RWR				GraphSAGE				PJsim			
Graph	AUC	P@K	AUPR	Acc.	AUC	P@K	AUPR	Acc.	AUC	P@K	AUPR	Acc.	AUC	P@K	AUPR	Acc.	AUC	P@K	AUPR	Acc.	AUC	P@K	AUPR	Acc.
G1	0.78	0.66	0.52	0.73	0.81	0.68	0.55	0.75	0.83	0.71	0.58	0.77	0.85	0.74	0.61	0.79	0.92	0.88	0.80	0.86	**0.90**	**0.86**	**0.78**	**0.84**
G2	0.75	0.62	0.48	0.70	0.78	0.65	0.50	0.72	0.80	0.68	0.53	0.74	0.83	0.71	0.56	0.76	0.91	0.87	0.79	0.85	**0.89**	**0.85**	**0.76**	**0.83**
G3	0.72	0.60	0.45	0.68	0.75	0.62	0.47	0.70	0.78	0.66	0.50	0.72	0.81	0.69	0.53	0.74	0.90	0.86	0.77	0.83	**0.88**	**0.84**	**0.75**	**0.81**
G4	0.70	0.58	0.43	0.66	0.73	0.60	0.45	0.68	0.76	0.64	0.48	0.70	0.80	0.67	0.51	0.73	0.89	0.85	0.76	0.82	**0.87**	**0.83**	**0.74**	**0.80**
G5	0.77	0.64	0.50	0.72	0.80	0.66	0.52	0.74	0.82	0.69	0.55	0.76	0.85	0.72	0.58	0.78	0.92	0.88	0.81	0.86	**0.90**	**0.86**	**0.79**	**0.84**
G6	0.76	0.63	0.49	0.71	0.79	0.65	0.51	0.73	0.81	0.68	0.54	0.75	0.84	0.71	0.57	0.77	0.91	0.87	0.80	0.85	**0.89**	**0.85**	**0.77**	**0.83**
G7	0.74	0.61	0.46	0.69	0.77	0.63	0.48	0.71	0.80	0.66	0.51	0.73	0.83	0.69	0.54	0.75	0.90	0.86	0.78	0.83	**0.88**	**0.84**	**0.76**	**0.81**
G8	0.73	0.60	0.45	0.68	0.76	0.62	0.47	0.70	0.79	0.65	0.50	0.72	0.82	0.68	0.53	0.74	0.89	0.85	0.77	0.82	**0.87**	**0.83**	**0.75**	**0.80**

broader range of research domains makes the semantic alignment task more meaningful, and PJsim is better able to exploit this structure. The numerically small absolute values observed across all DBLP datasets reflect the fine-grained and imbalanced nature of the label space, where many specialized research areas contain only a few nodes, making perfect top-k retrieval inherently challenging. On **DBLP_V1**, where the **label space is restricted to only two conference categories (CVPR and DBDM)**, all methods perform **relatively similarly**, though PJsim still leads with 0.213 (Kendall), 0.249 (Spearman), and 0.7957 (NDCG). *These results collectively demonstrate that PJsim maintains strong semantic consistency across settings, with its advantage becoming more prominent in semantically diverse environments.*

4.4.2 ***Link Prediction.*** To validate the applicability of PJsim for link prediction [3], we evaluated it on six graphs (G1 to G6 from Table 3) using the setup shown in Table 5. We follow a standard edge-recovery evaluation protocol in which 10% of the edges are removed uniformly at random from each graph. Each similarity method then scores all node pairs (u, v), and a removed edge is counted as correctly predicted if its similarity score is higher than that of randomly sampled non-edges. Accuracy is computed as the fraction of removed edges that satisfy this criterion. We compared PJsim against Cosine, Jaccard, SimRank (S_{Jeh}), and RWR [3], all of which are unsupervised similarity-based methods. GraphSAGE [15] serves as a supervised benchmark baseline that requires training on labeled data, providing an upper-bound reference for comparison. Four standard metrics were used: Area Under the ROC Curve (AUC), Accuracy, Area Under the Precision-Recall Curve (AUPR), and Precision at Top-K (P@K) (where K is 1000 in our experiments).

PJsim consistently outperformed all the unsupervised baselines and closely approached GraphSAGE. For instance, on G1, PJsim achieved an AUC of 0.90 and an accuracy of 0.84, compared to GraphSAGE's 0.92 and 0.86. Similar trends were observed across G2 to G6, where PJsim's Accuracy ranged from 0.81 to 0.84, with P@K reaching 0.85 or 0.86. Traditional heuristics like Cosine and Jaccard underperformed, with accuracy below 0.73 and AUPR often under 0.50, due to their shallow

Table 6. Link prediction performance under varying tolerance values. Columns report results for S_{Yu} and PJsim. Metrics include AUC (area under the curve), P@K (precision at top-K), AUPR (area under precision–recall), Acc. (accuracy), and Time (seconds).

		G1		G2		G3	
Tol.	Metric	S_{Yu}	PJsim	S_{Yu}	PJsim	S_{Yu}	PJsim
10^{-1}	AUC	0.81	**0.90**	0.78	**0.89**	0.77	**0.88**
	P@K	0.72	**0.86**	0.71	**0.85**	0.70	**0.84**
	AUPR	0.69	**0.78**	0.68	**0.78**	0.65	**0.75**
	Acc.	0.71	**0.84**	0.66	**0.83**	0.69	**0.81**
	Time	18.4	54.1	61.2	118.6	92.3	181.7
10^{-2}	AUC	0.85	**0.90**	0.82	**0.89**	0.80	**0.88**
	P@K	0.77	**0.86**	0.76	**0.85**	0.74	**0.84**
	AUPR	0.72	**0.78**	0.71	**0.78**	0.68	**0.75**
	Acc.	0.75	**0.84**	0.72	**0.83**	0.71	**0.81**
	Time	31.6	54.1	103.4	118.6	133.1	181.7
10^{-3}	AUC	0.87	**0.90**	0.85	**0.89**	0.83	**0.88**
	P@K	0.81	**0.86**	0.79	**0.85**	0.77	**0.84**
	AUPR	0.75	**0.78**	0.74	**0.78**	0.70	**0.75**
	Acc.	0.79	**0.84**	0.77	**0.83**	0.74	**0.81**
	Time	52.8	54.1	144.7	**118.6**	169.5	181.7
10^{-4}	AUC	0.88	**0.90**	0.87	**0.89**	0.85	**0.88**
	P@K	0.83	**0.86**	0.82	**0.85**	0.80	**0.84**
	AUPR	0.76	**0.78**	0.76	**0.78**	0.72	**0.75**
	Acc.	0.81	**0.84**	0.80	**0.83**	0.75	**0.81**
	Time	53.2	54.1	118.9	**118.6**	181.9	**181.7**
10^{-5}	AUC	0.89	**0.90**	0.88	**0.89**	0.87	**0.88**
	P@K	0.85	**0.86**	0.84	**0.85**	0.82	**0.84**
	AUPR	0.77	**0.78**	0.77	**0.78**	0.74	**0.75**
	Acc.	0.83	**0.84**	0.80	**0.83**	0.77	**0.81**
	Time	68.9	**54.1**	172.4	**118.6**	245.6	**181.7**
10^{-6}	AUC	0.90	**0.90**	0.89	**0.89**	0.88	**0.88**
	P@K	0.86	**0.86**	0.85	**0.85**	0.84	**0.84**
	AUPR	0.78	**0.78**	0.78	**0.78**	0.75	**0.75**
	Acc.	0.84	**0.84**	0.83	**0.83**	0.80	**0.81**
	Time	83.7	**54.1**	241.8	**118.6**	321.4	**181.7**

reliance on local neighborhoods. SimRank (S_{Jeh}) performed slightly better (accuracy 0.70–0.76), but often failed in sparse or asymmetric settings where it assigns zero similarity. PJsim addresses these limitations by accounting for all path-based contributions through a closed-form formulation, yielding robust similarity scores even in challenging topologies. On G4 and G5, PJsim achieved AUPR scores of 0.79 and an accuracy of 0.84, confirming its effectiveness for unsupervised link prediction and its ability to match supervised models without training overhead.

There may exist tolerance settings under which S_{Yu} achieves acceptable link prediction accuracy with reduced runtime. To examine this possibility, we evaluated link prediction for S_{Yu} and PJsim across tolerance values from 10^{-1} to 10^{-6} on graphs G1, G2, and G3; results for other graphs show similar trends and are omitted. Table 6 summarizes the results. Across all three graphs, S_{Yu} exhibits an accuracy–runtime trade-off. On G1, at tolerance 10^{-1}, S_{Yu} attains 0.81 AUC and 0.72 P@K in 18.4 seconds, compared to PJsim's 0.90 AUC and 0.86 P@K in 54.1 seconds. Tightening the tolerance to 10^{-4} improves S_{Yu}'s accuracy to 0.88 AUC and 0.83 P@K, with runtime increasing to 53.2 seconds, while still remaining below PJsim's accuracy. At tolerance 10^{-6}, S_{Yu} matches PJsim's accuracy but requires 83.7 seconds. A similar pattern is observed on G2 and G3. On G2, S_{Yu}'s runtime increases from 61.22 seconds at 10^{-1} to 241.8 seconds at 10^{-6}, exceeding PJsim's constant runtime of 118.6 seconds, while matching PJsim's accuracy only at the strictest tolerance. On G3, runtime increases from 92.32 seconds to 321.4 seconds, again exceeding PJsim's 181.7 seconds and

matching accuracy only at 10^{-6}. Overall, no tolerance setting allows S_{Yu} to simultaneously match PJsim in both accuracy and runtime.

4.5 Discussion

4.5.1 ***Effect of Graph Sparsity.*** PJsim shows its strongest performance gains on denser graphs, as evident from Figure 4. For example, on graphs such as G1, G4, G15, and G16, the speedups were the highest, reaching up to 3.22×. These graphs contain a larger number of edges, which makes iterative methods more expensive due to increased per-iteration cost and slower convergence. In contrast, PJsim benefits from efficient dense linear algebra kernels, such as matrix multiplication and Schur decomposition, which scale well with dense structures. On the other hand, for sparser graphs such as G2 and G10, the speedups were lower, although still consistently greater than 1×. In these cases, Yu's iterative formulation takes advantage of sparsity.

4.5.2 ***Effect of Graph Size.*** PJsim performs consistently better on large graphs as well. While the iterative method's cost depends on both the number of nodes and edges and grows rapidly with graph size due to repeated sparse multiplications over many iterations, PJsim's closed-form computation depends primarily on n. As a result, when the graph becomes larger and denser, the number of operations in the iterative method increases more sharply than in PJsim, leading to speedups for PJsim on large graphs. This trend is clearly visible in Figure 4, where larger real-world graphs show consistent speedup.

4.5.3 ***Spectral Structure and Numerical Properties.*** The performance of PJsim also depends on the spectral properties of the transition matrix Q. Graphs with well-conditioned matrices and clear spectral gaps enable stable and efficient Schur decomposition, which is central to PJsim's formulation. In such scenarios, PJsim achieves both accuracy (as validated in Section 4.3.2 and Table 4) and significant speedups. In contrast, for graphs with nearly reducible or poorly conditioned matrices, numerical stability issues may arise, and the decomposition step may take longer. Although these effects are less pronounced than sparsity or size, they can mildly reduce PJsim's advantage in certain cases.

5 Conclusion

In this work, we presented PJsim, an exact, closed-form solution for SimRank computation that leverages matrix diagonalization and the Kronecker product to solve the Lyapunov equation, ensuring robust, accurate similarity scores without reliance on convergence criteria. After exploring a Neumann Series approximation, we found it to be parameter-sensitive and computationally expensive, demonstrating that approximation does not always lead to efficiency gains. Our experiments show that $\text{PJsim}_{\text{EVD}}$ and PJsim_{BS} consistently outperform Yu's iterative formulation with up to 3.22× and 2.15× speedup respectively, while providing exact results, and case studies demonstrate their effectiveness for real-world applications, including link prediction and semantic fidelity evaluation, where they achieve higher accuracy than commonly used models.

Although PJsim effectively addresses the zero-similarity problem faster than iterative methods and performs reliably across diverse graph structures, its cubic runtime complexity can limit scalability for very large graphs, motivating future work to optimize computational complexity through parallelism on modern hardware such as many-core GPUs and extend the formulation to dynamic graphs for real-time similarity computation in evolving networks.

Acknowledgement

We appreciate the valuable comments provided by the reviewers. This work of Prajjwal Nijhara is supported by the TCS Research Scholar Program Cycle 19.

References

[1] Bjorn Adlerborn, Bo Kagstrom, and Daniel Kressner. 2014. A parallel QZ algorithm for distributed memory HPC systems. *SIAM Journal on Scientific Computing* 36, 5 (2014), C480–C503.

[2] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. 2008. Simrank++ query rewriting through link analysis of the clickgraph (poster). In *Proceedings of the 17th international conference on World Wide Web*. 1177–1178.

[3] Djihad Arrar, Nadjet Kamel, and Abdelaziz Lakhfif. 2024. A comprehensive survey of link prediction methods. *The journal of supercomputing* 80, 3 (2024), 3902–3942.

[4] Richard H Bartels and George W Stewart. 1972. Algorithm 432 [C2]: solution of the matrix equation AX+ XB= C [F4]. *Commun. ACM* 15, 9 (1972), 820–826.

[5] Peter Benner and Tobias Damm. 2011. Lyapunov equations, energy functionals, and model order reduction of bilinear and stochastic systems. *SIAM journal on control and optimization* 49, 2 (2011), 686–711.

[6] Yuanzhe Cai, Pei Li, Hongyan Liu, Jun He, and Xiaoyong Du. 2008. S-simrank: Combining content and link information to cluster papers effectively and efficiently. In *Advanced Data Mining and Applications: 4th International Conference, ADMA 2008, Chengdu, China, October 8-10, 2008. Proceedings 4*. Springer, 317–329.

[7] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. 2004. R-MAT: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 442–446.

[8] Hung-Hsuan Chen and C Lee Giles. 2015. Ascos++ an asymmetric similarity measure for weighted networks to address the problem of simrank. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 2 (2015), 1–26.

[9] Eberlein. 1987. On the Schur Decomposition of a Matrix for Parallel Computation. *IEEE Trans. Comput.* C-36, 2 (1987), 167–174. doi:10.1109/TC.1987.1676879

[10] Daniel Fogaras and Balazs Racz. 2007. Practical algorithms and lower bounds for similarity search in massive graphs. *IEEE transactions on knowledge and Data Engineering* 19, 5 (2007), 585–598.

[11] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, and Makoto Onizuka. 2013. Efficient search algorithm for SimRank. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 589–600.

[12] Zoran Gajic and Muhammad Tahir Javed Qureshi. 2008. *Lyapunov matrix equation in system stability and control*. Courier Corporation.

[13] Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2007. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Laboratory (LANL).

[14] Masoud Reyhani Hamedani and Sang-Wook Kim. 2016. SimRank and its variants in academic literature data: measures and evaluation. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 1102–1107.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[16] S. J. Hammerling. 1982. Numerical Solution of the Stable, Non-negative Definite Lyapunov Equation. *IMA J. Numer. Anal.* 2, 3 (07 1982), 303–323.

[17] Guoming He, Haijun Feng, Cuiping Li, and Hong Chen. 2010. Parallel SimRank computation on large graphs with iterative aggregation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 543–552.

[18] H.V. Henderson and S.R. Searle. 1980. The vec-permutation matrix, the vec operator and Kronecker products: A review. *Linear and Multilinear Algebra* 9, 4 (1980), 271–288.

[19] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 538–543.

[20] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Ke Xu. 2015. READS: a random walk approach for efficient and accurate dynamic SimRank. In *Proceedings of the VLDB Endowment*, Vol. 8. VLDB Endowment, 937–948.

[21] Bo Kågström and Daniel Kressner. 2007. Multishift Variants of the QZ Algorithm with Aggressive Early Deflation. *SIAM J. Matrix Anal. Appl.* 29, 1 (2007), 199–227.

[22] Pei Lee, Laks VS Lakshmanan, and Jeffrey Xu Yu. 2012. On top-k structural similarity search. In *2012 IEEE 28th international conference on data engineering*. IEEE, 774–785.

[23] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 177–187.

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

[25] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. 2010. Fast computation of simrank for static and dynamic information networks. In *Proceedings of the 13th International Conference on Extending Database Technology*. 465–476.

[26] Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. 2008. Accuracy estimate and optimization techniques for simrank computation. *Proceedings of the VLDB Endowment* 1, 1 (2008), 422–433.

[27] Mirko Myllykoski and Carl Christian Kjelgaard Mikkelsen. 2021. Task-based, GPU-accelerated and robust library for solving dense nonsymmetric eigenvalue problems. *Concurrency and Computation: Practice and Experience* 33, 11 (2021), e5915.
[28] NetworkX Developers. 2024. SimRank Similarity — NetworkX Documentation. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.similarity.simrank_similarity.html. Accessed: 2026-02-04.
[29] Mark Newman. 2018. *Networks*. Oxford university press.
[30] Prajjwal Nijhara, Jainan Tandel, Rohit Prajapati, and Dip Sankar Banerjee. 2026. A Precise and Closed-Form Solution for Edge-Ranking. In *Proceedings of the 27th International Conference on Distributed Computing and Networking*. 138–142.
[31] Arantxa Otegi, Eneko Agirre, and Paul Clough. 2014. Personalised PageRank for making recommendations in digital cultural heritage collections. In *IEEE/ACM Joint Conference on Digital Libraries*. IEEE, 49–52.
[32] Masoud Reyhani Hamedani, Sang-Wook Kim, Sang-Chul Lee, and Dong-Jin Kim. 2013. On exploiting content and citations together to compute similarity of scientific papers. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 1553–1556.
[33] Yingxia Shao, Bin Chen, Junjie Ma, and Bin Cui. 2014. Efficient top-k SimRank-based similarity join. In *Proceedings of the VLDB Endowment*, Vol. 7. VLDB Endowment, 847–858.
[34] Danny C Sorensen and Yunkai Zhou. 2003. Direct methods for matrix Sylvester and Lyapunov equations. *Journal of Applied Mathematics* 2003, 6 (2003), 277–303.
[35] Chang Sun, Bing-Quan Liu, Cheng-Jie Sun, De-Yuan Zhang, and Xiao-Long Wang. 2010. SimRank: A link analysis based blogger recommendation algorithm using text similarity. In *2010 International Conference on Machine Learning and Cybernetics*, Vol. 6. 3368–3373.
[36] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *Sixth international conference on data mining (ICDM'06)*. IEEE, 613–622.
[37] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2008. Random walk with restart: fast solutions and applications. *Knowledge and Information Systems* 14, 3 (2008), 327–346.
[38] Hanzhi Wang, Zhewei Wei, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Exact Single-Source SimRank Computation on Large Graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 653–663.
[39] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*. 1–8.
[40] Weiren Yu, Xuemin Lin, Wenjie Zhang, Lijun Chang, and Jian Pei. 2013. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *Proceedings of the VLDB Endowment* 7, 1 (2013), 13–24.
[41] Weiren Yu, Xuemin Lin, Wenjie Zhang, and Julie A McCann. 2014. Fast all-pairs SimRank assessment on large graphs and bipartite domains. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2014), 1810–1823.
[42] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A McCann. 2019. SimRank*: Effective and scalable pairwise similarity search based on graph topology. *The VLDB Journal* 28 (2019), 401–426.
[43] Weiren Yu and Julie A McCann. 2015. Efficient partial-pairs simrank search on large networks. *Proceedings of the VLDB Endowment* 8, 5 (2015), 569–580.
[44] Min Zhang and Weiren Yu. 2025. UPPR+: Scaling Uncertain Personalised PageRank Computation on Billion-Sized Graphs with Mutually Exclusive Edges. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2473–2482.
[45] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 553–562.
[46] Weiguo Zheng, Lei Zou, Yansong Feng, Lei Chen, and Dongyan Zhao. 2013. Efficient simrank-based similarity join over large graphs. *Proceedings of the VLDB Endowment* 6, 7 (2013), 493–504.
[47] Dengkui Zhu, Boyu Li, and Ping Liang. 2015. On the matrix inversion approximation based on Neumann series in massive MIMO systems. In *2015 IEEE international conference on communications (ICC)*. IEEE, 1763–1769.

Received October 2025; revised January 2026; accepted February 2026