Short Paper

GSim+: Efficient Retrieval of Node-to-Node Similarity Across Two Graphs at Billion Scale

Ruby Zhang
University of Warwick
Coventry, UK
ruby.zhang@warwick.ac.uk

Weiren Yu*
University of Warwick
Coventry, UK
weiren.yu@warwick.ac.uk

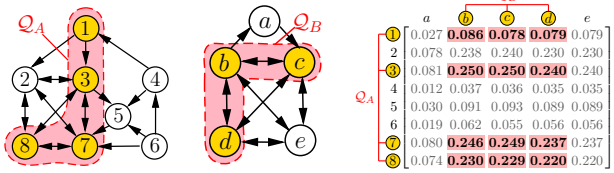
ABSTRACT

Numerous applications necessitate assessing the similarity between nodes across two graphs using link structures. GSim, suggested by Blondel *et al.*, is an appealing model, which determines the similarity between nodes across two graphs by recursively considering the similarity of their neighbouring structure. Regrettably, the existing efficient approach by Cason *et al.* using a low-rank approximation involves rather costly QR decomposition. Worse, a fixed low rank is relied on for similarity approximation throughout all iterations, leading to issues, *e.g.* overfitting or underestimation of the similarity matrix. In this paper, we focus on the efficient and scalable computation of GSim similarity on large graphs with provable guaranteed accuracy. First, we devise an efficient algorithm called GSim+, which resorts to an adaptive low-dimensional similarity matrix. GSim+ employs a novel iterative approach that dramatically speeds up the computation of GSim while yielding exactly the same scoring results of GSim at each iteration. Next, we theoretically provide an error bound on the iterative computation of GSim+, and analyse its computational complexity. Finally, we empirically validate the effectiveness of GSim+ to demonstrate that GSim+ is 1–4 orders of magnitude faster than state-of-the-art competitors with superior scalability on billion-edge graphs while guaranteeing the same accuracy of GSim at each iteration.

1 INTRODUCTION

A graph is an omnipresent data structure that describes intricate connections (edges) among objects (nodes). A fundamental task in graph mining is to quantify node-to-node similarity based on graph topology, known as *graph-based similarity search*. Recently, numerous appealing graph-based similarity measures have been proposed, *e.g.* VertexSim [13], GSim [4], SimRank [10], SimSem [15], CoSimRank [19], and SimRank$^{\#}$ [18]. Among them, GSim, conceived by Blondel *et al.* [4], is especially noteworthy for its simple and intuitive philosophy. The GSim similarity between nodes a and b across two graphs $\mathcal{G}_A$ and $\mathcal{G}_B$, denoted as $[\mathrm{S}]_{a,b}$, is determined by the similarity between the adjacent nodes of node a in $\mathcal{G}_A$ and the adjacent nodes of node b in $\mathcal{G}_B$. This idea allows GSim to be easily adapted to content-based similarity measures, making it a highly versatile model in a wide range of applications, *e.g.* synonym extraction [9] and web searching [16].

Similar to the SimRank measure [10] that follows the intuition that "two nodes in one graph are assessed as similar if their neighboring nodes are similar", the GSim model uses a recursive fashion to define node similarity, which progressively captures multi-hop neighbourly details pertaining to two given nodes across graphs. However, GSim also exhibits unique features when compared with SimRank: (1) For the base case of recursion, GSim initialises all pairs of nodes to a similarity of 1 ($[\mathrm{S}_0]_{a,b} \equiv 1,\ \forall(a,b)$), which is based on the philosophy that no other pair of nodes have a higher similarity than the pair itself. In contrast, SimRank initialises the similarity between any two distinct nodes as 0 ($[\mathrm{S}_0]_{a,b} = 0,\ \forall a \neq b$), which is based on the assumption that each node is always most similar to itself ($[\mathrm{S}_0]_{a,b} = 1,\ \forall a = b$). Recursively, due to the lack of connectivity between nodes across two graphs, SimRank would perceive these nodes as completely dissimilar, unlike GSim which would evaluate their similarity by recursively analysing their neighbouring patterns. (2) Unlike SimRank which requires the user to specify a damping factor, GSim does not rely on a damping factor.



(a) Facebook $\mathcal{G}_A$ (b) Twitter $\mathcal{G}_B$ (c) GSim Score $[\mathrm{S}]_{\mathcal{Q}_A,\mathcal{Q}_B}$

Figure 1: GSim Search over Two Query Sets $\mathcal{Q}_A$ and $\mathcal{Q}_B$ Across Two Graphs $\mathcal{G}_A$ (Facebook) and $\mathcal{G}_B$ (Twitter)

Application (Social Media Analysis). Consider two social networks in Figure 1, where $\mathcal{G}_A$ is a Facebook social media graph, and $\mathcal{G}_B$ is a Twitter micro-blogging graph. Each network consists of nodes denoting users, and edges depicting interactions (*e.g.* followers, likes). GSim similarity search over two query sets, $\mathcal{Q}_A$ and $\mathcal{Q}_B$ (in red), across two graphs can discover communities on Twitter that share similar interests or social communication patterns as communities on Facebook. This information is useful for many purposes (*e.g.* targeted advertising, or content recommendation), thereby enhancing marketing campaigns and user engagement. □

Despite its usefulness, the existing GSim algorithm [4] has two limitations. (1) It requires cubic time to perform matrix-matrix multiplications in each iteration. This high computational cost makes it impractical to deal with large graphs efficiently. (2) Even when the goal is to assess the similarity between a subset of nodes across two graphs, this method still requires the computation for all node pairs, as shown in Example 1.1.

Example 1.1. In Figure 1, given query sets $\mathcal{Q}_A = \{1,3,7,8\}$ and $\mathcal{Q}_B = \{b,c,d\}$ (in red), we want to assess GSim similarity $\{s(x,y)\}$ between any nodes $x \in \mathcal{Q}_A$ and $y \in \mathcal{Q}_B$ across graphs $\mathcal{G}_A$ and $\mathcal{G}_B$.

The naive approach [4] needs to calculate the pairwise similarities between all nodes across the two graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ and then extract the relevant queries. Specifically, we begin by initialising the similarity matrix as a matrix of all 1s. Then, we perform matrix multiplications to compute the entire similarity

*Weiren Yu is the corresponding author.

© 2024 Copyright held by the owner/author(s). Published in Proceedings of the 27th International Conference on Extending Database Technology (EDBT), 25th March-28th March, 2024, ISBN 978-3-89318-094-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

Algorithm	Time Complexity	Space Complexity	Notes
GSim+ (our)	$O(l(m_A + m_B + \|\mathcal{Q}_A\|\|\mathcal{Q}_B\|))$	$O(l(n_A + n_B))$	$l = \min\{2^k, n_A, n_B\}$
GSVD [6]	$O(r(m_A + m_B + n_A r + n_B r))$	$O(n_A n_B)$	r: fixed low rank of SVD
GSim [4]	$O(m_A n_B + m_B n_A)$	$O(n_A n_B)$	independent of query size
RSim [11]	$O(k(n_A + n_B)^2 d \log d)$	$O((n_A + n_B)^2)$	d: ave degree of $\mathcal{G}_A \cup \mathcal{G}_B$
NED [22]	$O(\|\mathcal{Q}_A\|\|\mathcal{Q}_B\|kL^3)$	$O(d^{k+1})$	L: # of nodes per level
SS-BC* [7]	$O(\|\mathcal{Q}_A\|\|\mathcal{Q}_B\|k \log D)$	$O(k(n_A + n_B) \log D)$	D: max degree of $\mathcal{G}_A \cup \mathcal{G}_B$

Table 1: Comparison of Various Algorithms for Similarity Search between Queries $\mathcal{Q}_A$ and $\mathcal{Q}_B$ across Graphs

matrix at the next iteration from the product of the two graph adjacency matrices and the similarity matrix at the current iteration. This iterative process goes on till the similarity matrix converges. Finally, we extract the elements indexed by queries $\mathcal{Q}_A$ and $\mathcal{Q}_B$ from the similarity matrix (red in Figure 1c). □

To accelerate the computation of GSim, the most efficient approach [6] utilizes a low-rank approximation technique. However, this method has the limitation of involving a costly QR decomposition. Moreover, it relies on a fixed low rank for approximating the similarity matrix for all iterations, as will be discussed in detail in Section 3. These limitations call for further improvements to enhance the efficiency and effectiveness of GSim.

Contributions. Our main contributions are as follows:

- We devise an efficient algorithm, GSim+, that resorts to an adaptive low-dimensional representation of GSim similarity matrix, which substantially accelerates the computation of GSim without any compromise in accuracy. (Section 3)
- We provide an error bound on the iterative computation of GSim+, and analyse the computational complexity of GSim+ in terms of both CPU time and memory. (Section 4)
- Using real datasets, we empirically validate that GSim+ surpasses its state-of-the-art competitors in speed by 1–4 orders of magnitude with high scalability on billion-edge graphs while ensuring the same accuracy as GSim. (Section 5)

Related Work. Recent years have witnessed a growing interest in graph similarity search. Representative models include GSim [4], GSVD [6], RoleSim (RSim) [11], NED [22], StructSim (SS-BC*) [7]. GSim is inspired by Kleinberg's HITS [12] that evaluates similarity from the graph dominant eigenvector. Blondel *et al.* [4] applied GSim for automatic synonym extraction in a dictionary. Bruch *et al.* [5] employed GSim to detect community in online dating Market structure analysis. However, GSim is rather expensive to compute via the naive fixed-point iteration [4], which hinders its scalability on large graphs. RoleSim (RSim) [11], is a role-based similarity model that ensures automorphism confirmation through maximal neighbor matching. It entails $O(k(n_A + n_B)^2 d \log d)$ time to assess similarities of all pairs in a graph $\mathcal{G}_A \cup \mathcal{G}_B$ with $(n_A + n_B)$ nodes and d average degree, which is not scalable well on large graphs. To accelerate computation further, IcebergRoleSim is proposed to evaluate only node pairs whose similarities exceed a specified threshold. However, this method does not improve the worst-case time complexity. NED [22] utilises the k-adjacent trees of two nodes to assess a single-pair similarity. It is more memory-efficient, only needing to store k-adjacent trees, unlike RoleSim, which demands significant memory for all-pair similarities in each iteration. However, the time complexity of NED is $O(KL^3)$ for querying a single pair, where L is the average number of nodes in each level of the k-adjacent trees. Unfortunately, due to the huge number of repeatedly visited nodes across multiple levels, L often exhibits exponential growth as k increases. Consequently, NED is exceedingly slow as k rises. StructSim (SS-BC*) [7] adopts a hierarchical framework to calculate a single-pair role similarity more efficiently using maximum matching. It incorporates BinCount matching to speed up computations in large-scale graphs. While SS-BC* excels in answering a single-pair similarity query, its time efficiency decreases when extended to handle $|\mathcal{Q}_A| \times |\mathcal{Q}_B|$ query pairs – SS-BC* requires the repeated execution of single-pair queries $|\mathcal{Q}_A| \times |\mathcal{Q}_B|$ times, resulting in many duplicate computations across queries, thus reducing its efficiency for partial-pair similarity searches. More recently, Cason *et al.* [6] improved the work of [8] and proposed a more efficient scheme GSVD, which employs low-rank SVD and QR decomposition to approximate the GSim matrix. However, this approach is limited by its fixed-rank iteration, which may lead to issues, *e.g.* overfitting or underestimation in similarity approximation. Moreover, the QR decomposition involved in their method is rather costly. There are also studies on optimising GSim-like models [10, 14, 17, 20, 21]. However, none of them exploits the low-rank characteristics of the evolving resolution of the GSim matrix.

Symbol	Description	Symbol	Description
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	graph ($\mathcal{V}$/$\mathcal{E}$: node/edge set)	$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$
$\mathbf{A}/\mathbf{B}$	adjacency matrix of $\mathcal{G}_A/\mathcal{G}_B$	$[\mathbf{X}]_{\mathcal{Q}_A,*}$	extract rows indexed by $\mathcal{Q}_A$ from $\mathbf{X}$
$\mathcal{Q}_A/\mathcal{Q}_B$	a set of query nodes in $\mathcal{G}_A/\mathcal{G}_B$	$\|\mathbf{X}\|_F$	Frobenius norm of matrix $\mathbf{X}$
$\mathbf{S}_k$	similarity matrix at iteration k	$\mathbf{Z}_k$	unnormalised similarity via Eq.(6a)
$n_A(m_A)$	number of nodes (edges) in $\mathcal{G}_A$	k	iteration number
$\vec{1}_n$	length-n column vector of all 1s	$\mathbf{U}_k/\mathbf{V}_k$	low-embeddings of $\mathbf{Z}_k$ via Eq.(8)–(9)

Table 2: Key Symbols and Their Meanings

2 PRELIMINARIES

In this section, we formally review the background of GSim. Table 2 lists the main symbols used throughout this paper.

Let $\mathcal{G}_A = (\mathcal{V}_A, \mathcal{E}_A)$ and $\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B)$ be two graphs with node sets $\mathcal{V}_A$ and $\mathcal{V}_B$, and edge sets $\mathcal{E}_A$ and $\mathcal{E}_B$. Let $\mathbf{A}$ and $\mathbf{B}$ be the adjacency matrices of $\mathcal{G}_A$ and $\mathcal{G}_B$, respectively. We denote by $n_A = |\mathcal{V}_A|$ and $n_B = |\mathcal{V}_B|$ the number of nodes in graph $\mathcal{G}_A$ and $\mathcal{G}_B$, respectively, and $m_A = |\mathcal{E}_A|$ and $m_B = |\mathcal{E}_B|$ the number of edges in $\mathcal{G}_A$ and $\mathcal{G}_B$, respectively. The GSim similarity matrix $\mathbf{S}$ $(n_A \times n_B)$ across graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ is defined recursively as

$$\mathbf{S} = \mathbf{A}\mathbf{S}\mathbf{B}^T + \mathbf{A}^T\mathbf{S}\mathbf{B} \tag{1}$$

where $(*)^T$ is matrix transpose. Each (i, j)-th entry, $[\mathbf{S}]_{i,j}$, measures GSim similarity between node i in $\mathcal{G}_A$ and node j in $\mathcal{G}_B$.

Blondel *et al.*'s GSim [4]. To obtain the solution $\mathbf{S}$ to Eq.(1), Blondel *et al.* [4] adopted the following power iteration method:

$$\mathbf{S}_k = \frac{\mathbf{A}\mathbf{S}_{k-1}\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_{k-1}\mathbf{B}}{\|\mathbf{A}\mathbf{S}_{k-1}\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_{k-1}\mathbf{B}\|_F} \ (k = 1, 2, \cdots) \ \text{with} \ \mathbf{S}_0 = \mathbf{1} \tag{2}$$

where $\mathbf{S}_k$ is the GSim similarity matrix at iteration k. Initially, $\mathbf{S}_0$ is set to $\mathbf{1}$ (a matrix of all 1s), meaning that there is no other pair of nodes that is more similar to each other than a pair itself. $\|*\|_F$ denotes a Frobenius norm, defined as the square root of the sum of the squares of the elements of the matrix. It has been shown in [4] that the even iterates of Eq.(2) converge to $\mathbf{S}$, *i.e.*, $\lim_{k\to\infty} \mathbf{S}_{2k} = \mathbf{S}$.

Cason *et al.*'s GSVD [6]. To speed up Eq.(1) computation, Cason *et al.* [6] proposed a low-rank approximation method. $\mathbf{S}_k$ is first approximated with $\mathbf{U}_k\Sigma_k\mathbf{V}_k^T$ via a fixed low rank-r SVD decomposition. To get the SVD approximation of $\mathbf{S}_{k+1}$ in the next iteration, they plug $\mathbf{S}_k \approx \mathbf{U}_k\Sigma_k\mathbf{V}_k^T$ into the RHS of Eq.(1) to get

$$\mathbf{A}\mathbf{S}_k\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_k\mathbf{B} \approx \left[\mathbf{A}\mathbf{U}_k\Sigma_k \mid \mathbf{A}^T\mathbf{U}_k\Sigma_k\right] \cdot \left[\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k\right]^T \tag{3}$$

Next, applying QR decomposition $\mathbf{Q}_\mathbf{U}\mathbf{R}_\mathbf{U}^T = \left[\mathbf{A}\mathbf{U}_k\Sigma_k \mid \mathbf{A}^T\mathbf{U}_k\Sigma_k\right]$ and $\mathbf{Q}_\mathbf{V}\mathbf{R}_\mathbf{V}^T = [\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k]$ plus another SVD decomposition

yields

$$\text{Eq.(3)} = \mathbf{Q}_\mathbf{U} \underbrace{(\mathbf{R}_\mathbf{U}\mathbf{R}_\mathbf{V}{}^T)}_{\approx\tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^T} \mathbf{Q}_\mathbf{V}{}^T \approx \underbrace{(\mathbf{Q}_\mathbf{U}\tilde{\mathbf{U}})}_{=\mathbf{U}_{k+1}} \underbrace{\tilde{\Sigma}}_{=\Sigma_{k+1}} \underbrace{(\mathbf{Q}_\mathbf{V}\tilde{\mathbf{V}})^T}_{=\mathbf{V}_{k+1}} = \mathbf{U}_{k+1}\Sigma_{k+1}\mathbf{V}_{k+1}^T \quad (4)$$

Thus, in the next iteration, $\mathbf{S}_{k+1}$ takes the similar form of $\mathbf{U}_{k+1}\Sigma_{k+1}\mathbf{V}_{k+1}^T$, just like $\mathbf{S}_k$ is represented as $\mathbf{U}_k\Sigma_k\mathbf{V}_k^T$ in the current iteration. This iterative process will continue until $\mathbf{S}_k$ converges.

3 OUR SOLUTION

Problem Statement. We aim to solve the following problem: *Given:* two graphs $\mathcal{G}_A$ and $\mathcal{G}_B$, and two query sets $\mathcal{Q}_A$ and $\mathcal{Q}_B$ *Retrieve:* the GSim similarities $[\mathbf{S}]_{\mathcal{Q}_A,\mathcal{Q}_B} := \{[\mathbf{S}]_{a,b}\}_{\forall(a,b)\in\mathcal{Q}_A\times\mathcal{Q}_B}$ between every node a in $\mathcal{G}_A$ and every node b in $\mathcal{G}_B$.

Limitations of Two State-of-The-Art Solutions ([4] and [6]). Blondel *et al.*'s GSim [4] has two limitations: Firstly, to compute $\mathbf{S}_k$ in Eq.(2), costly matrix multiplications ($\mathbf{A}\mathbf{S}_k\mathbf{B}^T$ and $\mathbf{A}^T\mathbf{S}_k\mathbf{B}$) are required per iteration, entailing $O(n_An_B(n_A+n_B))$ time. Secondly, even if we only want to assess partial entries of $\mathbf{S}_k$ (*e.g.* $[\mathbf{S}_k]_{\mathcal{Q}_A,\mathcal{Q}_B}$), Eq.(2) necessitates that all the $(n_A\times n_B)$ elements of $\mathbf{S}_{k-1}$ from the previous iteration be prepared first based on the following iteration:

$$\overbrace{[\mathbf{S}_{k+1}]_{\mathcal{Q}_A,\mathcal{Q}_B}}^{(|\mathcal{Q}_A|\times|\mathcal{Q}_B|)\text{ pairs}} = \frac{[\mathbf{A}]_{\mathcal{Q}_A,*}\overbrace{\mathbf{S}_k}^{(n_A\times n_B)\text{ pairs}}[\mathbf{B}^T]_{*,\mathcal{Q}_B}+[\mathbf{A}^T]_{\mathcal{Q}_A,*}\overbrace{\mathbf{S}_k}^{(n_A\times n_B)\text{ pairs}}[\mathbf{B}]_{*,\mathcal{Q}_B}}{\left\|[\mathbf{A}]_{\mathcal{Q}_A,*}\mathbf{S}_k[\mathbf{B}^T]_{*,\mathcal{Q}_B}+[\mathbf{A}^T]_{\mathcal{Q}_A,*}\mathbf{S}_k[\mathbf{B}]_{*,\mathcal{Q}_B}\right\|_F} \quad (5)$$

The limitations in Cason *et al.*'s GSVD [6] are two-fold. Firstly, it requires two QR decompositions of the matrices $[\mathbf{A}\mathbf{U}_k\Sigma_k \mid \mathbf{A}^T\mathbf{U}_k\Sigma_k]$ and $[\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k]$, respectively, via Eq.(4), resulting in dense matrices ($\mathbf{Q}_\mathbf{U}\mathbf{R}_\mathbf{U}$) and ($\mathbf{Q}_\mathbf{V}\mathbf{R}_\mathbf{V}$). This process is very time-consuming. Secondly, approximating $[\mathbf{A}\mathbf{U}_k\Sigma_k \mid \mathbf{A}^T\mathbf{U}_k\Sigma_k]$ and $[\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k]$ in Eq.(3) with a fixed rank-r SVD may lead to overfitting or underestimation of the similarity matrix. This is because the dimensions of these block matrices vary as k rises. When k is large, approximating the large block matrices $[\mathbf{A}\mathbf{U}_k\Sigma_k \mid \mathbf{A}^T\mathbf{U}_k\Sigma_k]$ and $[\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k]$ in Eq.(3) with a small rank-r SVD may lead to a huge approximation error. Hence, it is undesirable to use a constant rank-r approximation of $\mathbf{S}_k$ for all iterations.

Our Scheme. We first provide an equivalent form of Eq.(2):

$$\mathbf{Z}_k = \mathbf{A}\mathbf{Z}_{k-1}\mathbf{B}^T + \mathbf{A}^T\mathbf{Z}_{k-1}\mathbf{B} \;\; (k=1,\cdots,K) \;\text{ with }\; \mathbf{Z}_0 = \mathbf{1} \quad (6a)$$

$$\mathbf{S}_K = \mathbf{Z}_K/\|\mathbf{Z}_K\|_F \;\; (\text{normalisation only in last iteration } K) \quad (6b)$$

It can be readily verified that, given the number of iterations, K, the solution $\mathbf{S}_K$ to Eq.(2) is exactly the same as the solution $\mathbf{S}_K$ to Eq.(6b). However, unlike Eq.(2), which mandates $\|*\|_F$ normalisation for every iteration, Eq.(6) only requires normalisation of $\mathbf{Z}_k$ to be performed once at the last iteration K. Due to this equivalence, our techniques below will focus on optimising $\mathbf{Z}_k$ in Eq.(6a).

Our key insight is that the rank of the matrix $\mathbf{Z}_k$ grows gently from 1 by a factor of 2 as Eq.(6a) iterates. Therefore, rather than computing the entire $\mathbf{Z}_k$ at each iteration, we progressively maintain two low-embedding matrices[1] for $\mathbf{Z}_k$ such that $\mathbf{Z}_k$ can be expressed as their outer product. At each iteration, we utilise the matrix associative law to parenthesise the matrix product with these low-embedding matrices that have adaptive ranks and dimensions to optimise $\mathbf{Z}_k$ computation, instead of representing $\mathbf{Z}_k$ with a constant rank-r SVD across all iterations, as in [6]. By dynamically maintaining the two low-embeddings for $\mathbf{Z}_k$ on an as-needed basis, our method avoids the overfitting or underestimation issue in [6].

Specifically, initially when $k=0$, $\mathbf{Z}_0$ is a rank-1 matrix of all 1s ($n_A\times n_B$), expressible as the outer product of two vectors of 1s:

$$\mathbf{Z}_0 = \vec{\mathbf{1}}_{n_A}\cdot\vec{\mathbf{1}}_{n_B}^T \quad\text{with}\quad \vec{\mathbf{1}}_{n_A} = [1,1,\cdots,1]^T \in \mathbb{R}^{n_A\times 1}$$

By virtue of the two low-rank embeddings, $\vec{\mathbf{1}}_{n_A}$ and $\vec{\mathbf{1}}_{n_B}$, for $\mathbf{Z}_0$, the computation of $\mathbf{Z}_1$ can be expedited considerably through the matrix associative law and parenthesisation, as shown below:

$$\begin{aligned}\mathbf{Z}_1 &= \mathbf{A}\mathbf{Z}_0\mathbf{B}^T + \mathbf{A}^T\mathbf{Z}_0\mathbf{B} = \mathbf{A}(\vec{\mathbf{1}}_{n_A}\vec{\mathbf{1}}_{n_B}^T)\mathbf{B}^T + \mathbf{A}^T(\vec{\mathbf{1}}_{n_A}\vec{\mathbf{1}}_{n_B}^T)\mathbf{B}\\ &= (\mathbf{A}\vec{\mathbf{1}}_{n_A})(\mathbf{B}\vec{\mathbf{1}}_{n_B})^T + (\mathbf{A}^T\vec{\mathbf{1}}_{n_A})(\mathbf{B}^T\vec{\mathbf{1}}_{n_B})^T\\ &= \underbrace{[\mathbf{A}\vec{\mathbf{1}}_{n_A}|\mathbf{A}^T\vec{\mathbf{1}}_{n_A}]}_{=\mathbf{U}_1}\underbrace{[\mathbf{B}\vec{\mathbf{1}}_{n_B}|\mathbf{B}^T\vec{\mathbf{1}}_{n_B}]^T}_{=\mathbf{V}_1} = \mathbf{U}_1{\mathbf{V}_1}^T\end{aligned} \quad (7)$$

It is noteworthy that, in the first iteration, our method by Eq.(7) enables a significant reduction in computational time, as opposed to the naive approach that requires $O(m_An_B+m_Bn_A)$ to perform matrix-matrix products (*e.g.* $\mathbf{A}\mathbf{Z}_0\mathbf{B}^T$ and $\mathbf{A}^T\mathbf{Z}_0\mathbf{B}$). Instead, our approach maintains two low-rank embedding matrices, $\mathbf{U}_1$ and $\mathbf{V}_1$, for the similarity $\mathbf{Z}_1$, resulting in a computational time of only $O(2(m_A+m_B))$. This significant reduction is achieved by leveraging the adaptive low-rank representation of $\mathbf{Z}_0$ and matrix associative law for parenthesisation, which only entails $O(2m_A)$ (*resp.* $O(2m_B)$) time to perform matrix-vector products $\mathbf{A}\vec{\mathbf{1}}_{n_A}$ and $\mathbf{A}^T\vec{\mathbf{1}}_{n_B}$ (*resp.* $\mathbf{B}\vec{\mathbf{1}}_{n_B}$ and $\mathbf{B}^T\vec{\mathbf{1}}_{n_B}$). Furthermore, as observed from Eq.(7), the resulting matrix $\mathbf{Z}_1$ has at most rank-2, due to the existence of two slender matrices $\mathbf{U}_1$ ($n_A\times 2$) and $\mathbf{V}_1$ ($n_B\times 2$) such that $\mathbf{Z}_1 = \mathbf{U}_1{\mathbf{V}_1}^T$. The iterative process continues till k converges. For each iteration k, it suffices to maintain only the low-embedding matrices, $\mathbf{U}_k$ and $\mathbf{V}_k$, for $\mathbf{Z}_k$. Only in the last iteration K, the outer product of $\mathbf{U}_K$ and $\mathbf{V}_K$ is carried out to obtain $\mathbf{S}_K$.

Precisely, we have the following theorem, which shows that our ideas yield identical results to the GSim similarity in Eq.(2).

Theorem 3.1. *Let* $\mathbf{U}_k$ *and* $\mathbf{V}_k$ *be slender matrices defined as*

$$\mathbf{U}_k = [\mathbf{A}\mathbf{U}_{k-1} \mid \mathbf{A}^T\mathbf{U}_{k-1}] \quad (k=1,2,\cdots) \;\; \textit{with} \;\; \mathbf{U}_0 = \vec{\mathbf{1}}_{n_A} \quad (8)$$

$$\mathbf{V}_k = [\mathbf{B}\mathbf{V}_{k-1} \mid \mathbf{B}^T\mathbf{V}_{k-1}] \quad (k=1,2,\cdots) \;\; \textit{with} \;\; \mathbf{V}_0 = \vec{\mathbf{1}}_{n_B} \quad (9)$$

The GSim similarity $\mathbf{S}_k$ *in Eq.*(2) *can be represented as*

$$\mathbf{S}_k = \mathbf{U}_k\mathbf{V}_k^T/\|\mathbf{U}_k\mathbf{V}_k^T\|_F \quad (10)$$

Proof. We prove Eq.(10) by induction on k.

Base Case: For $k=1$, $\mathbf{U}_1 = [\mathbf{A}\vec{\mathbf{1}}_{n_A} \mid \mathbf{A}^T\vec{\mathbf{1}}_{n_A}]$, $\mathbf{V}_1 = [\mathbf{B}\vec{\mathbf{1}}_{n_B} \mid \mathbf{B}^T\vec{\mathbf{1}}_{n_B}]$.

$$\mathbf{U}_1\mathbf{V}_1^T = \mathbf{A}\cdot\vec{\mathbf{1}}_{n_A}\vec{\mathbf{1}}_{n_B}^T\cdot\mathbf{B}^T + \mathbf{A}^T\cdot\vec{\mathbf{1}}_{n_A}\vec{\mathbf{1}}_{n_B}^T\cdot\mathbf{B} = \mathbf{A}\mathbf{S}_0\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_0\mathbf{B}$$

Thus, $\mathbf{S}_1 = \mathbf{U}_1\mathbf{V}_1^T/\|\mathbf{U}_1\mathbf{V}_1^T\|_F$, and Eq.(10) holds for $k=1$.

Induction Step: Assume the induction hypothesis $\mathbf{S}_k = \mathbf{U}_k\mathbf{V}_k^T/\|\mathbf{U}_k\mathbf{V}_k^T\|_F$ holds for k. We can deduce that

$$\begin{aligned}\mathbf{U}_{k+1}\mathbf{V}_{k+1}^T &= [\mathbf{A}\mathbf{U}_k \mid \mathbf{A}^T\mathbf{U}_k]\cdot[\mathbf{B}\mathbf{V}_k \mid \mathbf{B}^T\mathbf{V}_k]^T\\ &= \mathbf{A}(\mathbf{U}_k\mathbf{V}_k^T)\mathbf{B}^T + \mathbf{A}^T(\mathbf{U}_k\mathbf{V}_k^T)\mathbf{B}\end{aligned}$$

which implies that

$$\frac{\mathbf{U}_{k+1}\mathbf{V}_{k+1}^T}{\left\|\mathbf{U}_{k+1}\mathbf{V}_{k+1}^T\right\|_F} = \frac{\mathbf{A}\Big(\overbrace{\frac{\mathbf{U}_k\mathbf{V}_k^T}{\|\mathbf{U}_k\mathbf{V}_k^T\|_F}}^{\mathbf{S}_k}\Big)\mathbf{B}^T + \mathbf{A}^T\Big(\overbrace{\frac{\mathbf{U}_k\mathbf{V}_k^T}{\|\mathbf{U}_k\mathbf{V}_k^T\|_F}}^{\mathbf{S}_k}\Big)\mathbf{B}}{\left\|\mathbf{A}\Big(\frac{\mathbf{U}_k\mathbf{V}_k^T}{\|\mathbf{U}_k\mathbf{V}_k^T\|_F}\Big)\mathbf{B}^T + \mathbf{A}^T\Big(\frac{\mathbf{U}_k\mathbf{V}_k^T}{\|\mathbf{U}_k\mathbf{V}_k^T\|_F}\Big)\mathbf{B}\right\|_F} = \mathbf{S}_{k+1}$$

Thus, Eq.(10) holds for $k+1$, which completes the induction. □

[1] The low-embeddings for a given matrix $\mathbf{X}$ ($n\times m$) refers to two low-dimensional matrices $\mathbf{A}$ ($n\times r$) and $\mathbf{B}$ ($r\times m$) such that $\mathbf{A}\cdot\mathbf{B}=\mathbf{X}$, where $r\ll\min\{n,m\}$.

GSim+ Algorithm. Theorem 3.1 implies an efficient algorithm, denoted as GSim+, for similarity computation (see Algorithm 1).

Algorithm 1: GSim+ $(\mathbf{A}, \mathbf{B}, K, Q_A, Q_B)$

Input : $\mathbf{A}/\mathbf{B}$: adjacency matrix of $\mathcal{G}_A(\mathcal{V}_A, \mathcal{E}_A)/\mathcal{G}_B(\mathcal{V}_B, \mathcal{E}_B)$,
Q_A/Q_B: a set of queries in graph $\mathcal{V}_A/\mathcal{V}_B$,
K: total number of iterations.

Output: $\mathbf{S}_K$: pairwise similarity matrix ($|Q_A| \times |Q_B|$) between queries Q_A and Q_B across graphs at iteration K.

1 initialise $\mathbf{U}_0 := \vec{\mathbf{1}}_{n_A}$ and $\mathbf{V}_0 := \vec{\mathbf{1}}_{n_B}$
2 **for** $k := 1, 2, \cdots, K$ **do**
3 compute $\boldsymbol{\xi}_1 := \mathbf{A}\mathbf{U}_{k-1}$ and $\boldsymbol{\xi}_2 := \mathbf{A}^T\mathbf{U}_{k-1}$
4 compute $\boldsymbol{\eta}_1 := \mathbf{B}\mathbf{V}_{k-1}$ and $\boldsymbol{\eta}_2 := \mathbf{B}^T\mathbf{V}_{k-1}$
5 update $\mathbf{U}_k := [\boldsymbol{\xi}_1 \mid \boldsymbol{\xi}_2]$ and $\mathbf{V}_k := [\boldsymbol{\eta}_1 \mid \boldsymbol{\eta}_2]$
6 $\mathbf{Z} := [\mathbf{U}_K]_{Q_A,*}([\mathbf{V}_K]_{Q_B,*})^T$
7 **return** $\mathbf{S}_K := \mathbf{Z}/\|\mathbf{Z}\|_F$

Example 3.2. Recall Figure 1. Given $K = 2$, $Q_A = \{1, 3, 7, 8\}$, and $Q_B = \{b, c, d\}$, GSim+ computes $[\mathbf{S}_2]_{Q_A,Q_B}$ as follows:

First, $\mathbf{U}_0$ and $\mathbf{V}_0$ are initialised as two vectors of all 1s (line 1). Then, $\mathbf{U}_k$ and $\mathbf{V}_k$ ($k = 1, 2$) are iteratively evaluated (lines 2–5) as follows:

k	$\mathbf{U}_k$	$\mathbf{V}_k$
0	$\begin{bmatrix}1&1&1&1&1&1&1&1\end{bmatrix}^T$	$\begin{bmatrix}1&1&1&1&1\end{bmatrix}^T$
1	$\begin{bmatrix}2&3&4&2&1&3&3&3\\1&4&4&1&3&0&5&3\end{bmatrix}^T$	$\begin{bmatrix}1&4&3&3&3\\1&3&4&3&3\end{bmatrix}^T$
2	$\begin{bmatrix}7&10&10&3&3&6&10&10\\8&12&15&4&5&9&11&13\\2&12&11&3&9&0&14&10\\1&13&13&0&5&0&14&13\end{bmatrix}^T$	$\begin{bmatrix}3&10&10&10&10\\4&11&9&10&10\\4&9&11&10&10\\3&10&10&10&10\end{bmatrix}^T$

Next, we extract necessary rows from $\mathbf{U}_2$ and $\mathbf{V}_2$, respectively, based on query sets Q_A and Q_B to compute $\mathbf{Z}$ (line 6), which yields

$$\underset{[\mathbf{U}_2]_{Q_A,*}}{\begin{bmatrix}7&8&2&1\\10&15&11&13\\10&11&14&14\\10&13&10&13\end{bmatrix}} \underset{[\mathbf{V}_2]^T_{Q_B,*}}{\begin{bmatrix}10&10&10\\11&9&10\\9&11&10\\10&10&10\end{bmatrix}} = \underset{\mathbf{Z}}{\begin{bmatrix}186&174&180\\494&486&490\\487&493&490\\463&457&460\end{bmatrix}} \Longrightarrow \mathbf{S}_2 = \underset{\mathbf{Z}/\|\mathbf{Z}\|_F}{\begin{bmatrix}0.126&0.118&0.122\\0.335&0.330&0.332\\0.330&0.335&0.332\\0.314&0.310&0.312\end{bmatrix}}$$

Finally, using $\|\mathbf{Z}\|_F = 1474$, we normalise $\mathbf{Z}$ to obtain $\mathbf{S}_2$ (line 7). □

4 PERFORMANCE ANALYSIS OF GSIM+

We first analyse the time and space complexity of GSim+ (Theorem 4.1). We next provide an error bound for GSim+ (Theorem 4.2).

Theorem 4.1 (Time and Space Complexity). *GSim+ requires $O(l(m_A+m_B+|Q_A||Q_B|))$ time and $O(l(n_A+n_B))$ memory space (where $l = \min\{2^K, n_A, n_B\}$) to retrieve similarity $\mathbf{S}_K$ between two query sets Q_A and Q_B across graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ after K iterations.*

Proof. In line 1, it requires $O(n_A)$ (*resp.* $O(n_B)$) time and space to initialise $\mathbf{U}_0$ (*resp.* $\mathbf{V}_0$). At each k-th iteration (lines 2–5), it takes $O(2^{k-1}m_A)$ time and $O(2^{k-1}n_A + m_A)$ space to compute $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ (line 3). Similarly, it takes $O(2^{k-1}m_B)$ time and $O(2^{k-1}n_B + m_B)$ space to compute $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$ (line 4). Merging $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2$ (*resp.* $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$) into a block matrix $\mathbf{U}_k$ (*resp.* $\mathbf{V}_k$) requires $O(n_A 2^{k-1})$ (*resp.* $O(n_B 2^{k-1})$) time and space (line 5). Thus, for K iterations, it entails $O(\sum_{k=1}^{K} 2^{k-1}(m_A + m_B) = O(2^K(m_A+m_B))$ time to obtain $\mathbf{U}_K$ and $\mathbf{V}_K$. Next, it takes $O(2^K|Q_A||Q_B|)$ time and $O((|Q_A| + |Q_B|)2^K + |Q_A||Q_B|)$ space to compute $\mathbf{Z}$ (line 6). Finally, normalising $\mathbf{Z}$ requires $O(|Q_A||Q_B|)$ time and space (line 7). Thus, the total complexity of Algorithm 1 is $O(2^K(m_A + m_B + |Q_A||Q_B|))$ time and $O(2^K(n_A + n_B))$ space. Since for large graphs, $n_A \gg 2^K$ and $n_B \gg 2^K$, the term 2^K can be replaced with $l = \min\{2^K, n_A, n_B\}$. □

Theorem 4.2 (Error Bound). *Let $\mathbf{S}_k$ be the similarity matrix produced by GSim+ after k iterations, and $\mathbf{S}$ be the exact solution to Eq.(1). For any even number of iterations k, the gap between $\mathbf{S}_k$ and $\mathbf{S}$ is bounded by*

$$\|\mathbf{S}_k - \mathbf{S}\|_F \le \left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k C \quad with \quad C = \frac{1}{|c_1|}\sqrt{\textstyle\sum_{i=2}^{n} {c_i}^2} \tag{11}$$

where λ_1 and λ_2 are the largest and second largest eigenvalues of the matrix $\mathbf{M} = \mathbf{B} \otimes \mathbf{A} + \mathbf{B}^T \otimes \mathbf{A}^T$, respectively. $n = n_A \times n_B$. Each scalar c_i is the i-th element of the length-n column vector $\vec{\mathbf{c}} = [c_1, c_2, \cdots, c_n]^T$ which is derived from $\vec{\mathbf{c}} = \mathbf{V}^T\vec{\mathbf{1}}_n$ where $\mathbf{V}$ is a matrix whose columns are the corresponding eigenvectors of the matrix $\mathbf{M}$.

Proof. By Theorem 3.1, $\mathbf{S}_k$ in Eq.(10) can be expressed as

$$\mathbf{S}_k = \frac{\mathbf{U}_k\mathbf{V}_k^T}{\|\mathbf{U}_k\mathbf{V}_k^T\|_F} = \frac{\mathbf{A}\mathbf{S}_{k-1}\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_{k-1}\mathbf{B}}{\|\mathbf{A}\mathbf{S}_{k-1}\mathbf{B}^T + \mathbf{A}^T\mathbf{S}_{k-1}\mathbf{B}\|_F} \tag{12}$$

Let $\vec{\mathbf{s}}_k = vec(\mathbf{S}_k)$, $\mathbf{M} = \mathbf{B} \otimes \mathbf{A} + (\mathbf{B} \otimes \mathbf{A})^T$. Taking $vec(*)$ on both sides of Eq.(12), and applying the property of the Kronecker product $vec(\mathbf{A}\mathbf{X}\mathbf{B}^T) = (\mathbf{B} \otimes \mathbf{A})vec(\mathbf{X})$ produces

$$\vec{\mathbf{s}}_k = \frac{\mathbf{M}\vec{\mathbf{s}}_{k-1}}{\|\mathbf{M}\vec{\mathbf{s}}_{k-1}\|_2} = \cdots = \frac{\mathbf{M}^k\vec{\mathbf{s}}_0}{\|\mathbf{M}^k\vec{\mathbf{s}}_0\|_2} = \frac{\mathbf{M}^k\vec{\mathbf{1}}_n}{\|\mathbf{M}^k\vec{\mathbf{1}}_n\|_2} \quad (n = n_A \times n_B) \tag{13}$$

Let $\mathbf{W}$ be the eigenvectors of $\mathbf{M}$ associated with the eigenvalues $\boldsymbol{\Lambda}$:

$$\mathbf{M} = \mathbf{W}\boldsymbol{\Lambda}\mathbf{W}^T \quad \text{and} \quad \mathbf{M}^k = \mathbf{W}\boldsymbol{\Lambda}^k\mathbf{W}^T \tag{14}$$

Plugging Eq.(14) into Eq.(13) produces

$$\|\mathbf{M}^k\vec{\mathbf{1}}_n\|_2^2 = \vec{\mathbf{1}}_n^T\mathbf{M}^k \cdot \mathbf{M}^k\vec{\mathbf{1}}_n = (\vec{\mathbf{1}}_n^T\mathbf{W})\boldsymbol{\Lambda}^{2k}(\mathbf{W}^T\vec{\mathbf{1}}_n)$$

Let $\vec{\mathbf{c}} = [c_1, c_2, \cdots, c_n]^T \triangleq \mathbf{W}^T\vec{\mathbf{1}}_n$ and $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$. Assume the eigenvalues of $\mathbf{M}$ are sorted as $|\lambda_1| \ge \cdots \ge |\lambda_n|$. Then,

$$\|\mathbf{M}^k\vec{\mathbf{1}}_n\|_2 = \sqrt{\vec{\mathbf{c}}^T\boldsymbol{\Lambda}^{2k}\vec{\mathbf{c}}} = \sqrt{\textstyle\sum_{i=1}^{n} {\lambda_i}^{2k}{c_i}^2} \ge \sqrt{{\lambda_1}^{2k}{c_1}^2} = |\lambda_1|^k|c_1|$$

Substituting $\|\mathbf{M}^k\vec{\mathbf{1}}_n\|_2 = |\lambda_1|^k|c_1|$ back into Eq.(13) yields

$$\vec{\mathbf{s}}_k = \mathbf{M}^k\vec{\mathbf{1}}_n/\|\mathbf{M}^k\vec{\mathbf{1}}_n\|_2 \le \mathbf{M}^k\vec{\mathbf{1}}_n/(|\lambda_1|^k\,|c_1|) \tag{15}$$

Similarly, plugging Eq.(14), we have $\mathbf{M}^k\vec{\mathbf{1}}_n = \mathbf{W}\boldsymbol{\Lambda}^k\mathbf{W}^T\vec{\mathbf{1}}_n = \mathbf{W}\boldsymbol{\Lambda}^k\vec{\mathbf{c}}$ with $\vec{\mathbf{c}} = \mathbf{W}^T\vec{\mathbf{1}}_n$. Let $\mathbf{W} = [\mathbf{w}_1|\cdots|\mathbf{w}_n]$, where $\mathbf{w}_i$ is a vector. Then,

$$\mathbf{M}^k\vec{\mathbf{1}}_n = \sum_{i=1}^{n} c_i\lambda_i^k\mathbf{w}_i = c_1\lambda_1^k\mathbf{w}_1 + \sum_{i=2}^{n} c_i\lambda_i^k\mathbf{w}_i \tag{16}$$

Since $\lambda_1^k = |\lambda_1|^k$ for even number k, we plug Eq.(16) into Eq.(15):

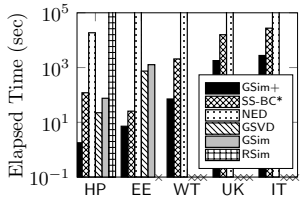
$$\vec{\mathbf{s}}_k \le \frac{c_1}{|c_1|}\mathbf{w}_1 + \sum_{i=2}^{n} \frac{c_i}{|c_1|}\frac{\lambda_i^k}{|\lambda_1|^k}\mathbf{w}_i \tag{17}$$

Next, we plug Eq.(16) into Eq.(14), and then take the limit on both sides as $k \to \infty$, which produces $\vec{\mathbf{s}} = (c_1/|c_1|)\mathbf{w}_1$. Finally, we subtract Eq.(17) from $\vec{\mathbf{s}}$, take 2-norm on both sides, and use the matrix norm property that $\|vec(\mathbf{X})\|_2 = \|\mathbf{X}\|_F$, which yields

$$\|\mathbf{S}_k - \mathbf{S}\|_F = \|\vec{\mathbf{s}}_k - \vec{\mathbf{s}}\|_2 \le \left\|\sum_{i=2}^{n} \frac{c_i\lambda_i^k}{|c_1||\lambda_1|^k}\mathbf{w}_i\right\|_2 = \sqrt{\sum_{i=2}^{n}\left(\frac{c_i\lambda_i^k}{|c_1||\lambda_1|^k}\right)^2 \mathbf{w}_i^T\mathbf{w}_i}$$

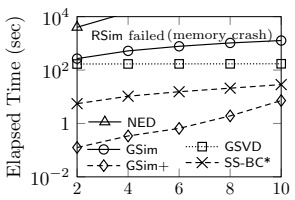
$$\le \sqrt{\sum_{i=2}^{n}\left(\frac{c_i\lambda_2^k}{|c_1||\lambda_1|^k}\right)^2} = \left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k C \quad \text{with} \quad C = \frac{1}{|c_1|}\sqrt{\textstyle\sum_{i=2}^{n} {c_i}^2}$$

□

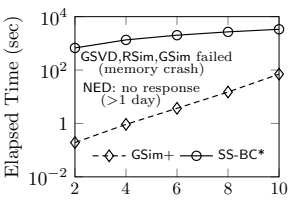


(a) Vary $\mathcal{G}_A$ on Real Datasets

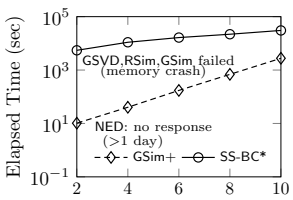
Figure 2: Time & Scalability



(a) Vary k on EE

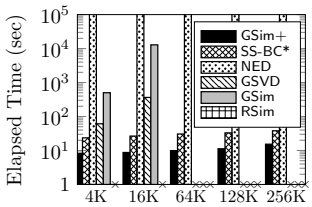


(b) Vary k on WT

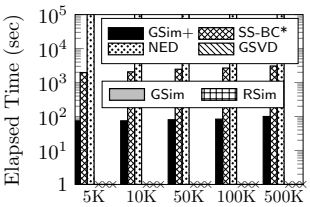


(c) Vary k on IT

Figure 3: Effect of k on Time over Real Datasets (EE, WT, IT)

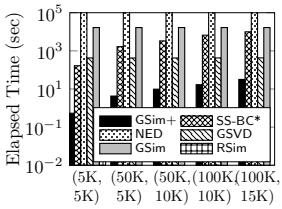


(a) Vary $|\mathcal{V}_B|$ on EE

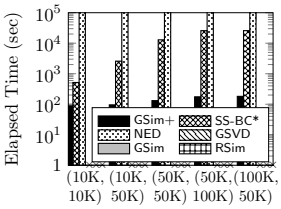


(b) Vary $|\mathcal{V}_B|$ on WT

Figure 4: Effect of $|\mathcal{V}_B|$ on Time



(a) Vary $(|Q_A|, |Q_B|)$ on EE



(b) Vary $(|Q_A|, |Q_B|)$ on WT

Figure 5: Effect of $(|Q_A|, |Q_B|)$ on Time

5 EXPERIMENTS

5.1 Experimental Settings

Datasets. We use real-life datasets publicly available at [1–3].

$\mathcal{G}_A$	$m = \lvert\mathcal{E}_A\rvert$	$n = \lvert\mathcal{V}_A\rvert$	m/n	**Description**
HP	421,578	34,546	12.2	Social friendship from ego-Facebook [3]
EE	420,045	265,214	1.6	A EU research institution Email network [3]
WT	5,021,410	2,394,385	2.1	Wikipedia talk (communication) graph [3]
UK	298,113,762	18,520,486	16.1	2002 large web crawl of **.uk** domain [2]
IT	1,150,725,436	41,291,594	27.9	2004 large web crawl of **.it** domain [2]

$\mathcal{G}_A$ is directly taken from these datasets, whereas the corresponding $\mathcal{G}_B$ are sampled from a subgraph of $\mathcal{G}_A$ with the size $|\mathcal{V}_B| = 10,000$.

Algorithms: We compare the following rivals:

- GSim+: our proposed scheme in Algorithm 1.
- GSVD [6]: the best-known competitor via low-rank SVD.
- GSim [4]: the conventional algorithm via power iteration.
- SS-BC* [7]: the StructSim hierarchical framework to calculate a single-pair role similarity through BinCount matching.
- NED [22]: a single-pair role similarity model that utilises the k-adjacent trees of two nodes to assess the similarity.
- RSim [11], the RoleSim similarity model that ensures automorphism confirmation via maximal neighbor matching.

Default Parameters: Unless otherwise specified, the number of iterations $k = 10$. The query size $|Q_A| = 2,000$ and $|Q_B| = 2,000$ (*resp.* $|Q_B| = 20,000$) for large WT, UK, and IT.

All experiments were conducted on Rocky Linux 8.7, using an Intel Core Xeon CPU E5-2660 v3 @ 2.60GHz, with 256GB RAM.

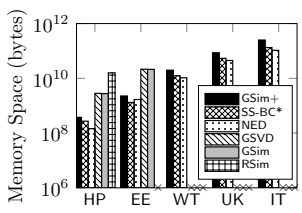
5.2 Experimental Results

5.2.1 Time Efficiency. The first set of experiments is to evaluate the time and scalability of GSim+ against its rivals.

Varying $\mathcal{G}_A$ on Real Datasets. Figure 2 compares the time of GSim+ with its competitors on five real graphs. 1) On each dataset, GSim+ consistently outperforms other algorithms, especially with larger graphs (UK and IT). This is because GSim+ can avoid costly matrix-matrix multiplications for similarity assessment through low-embeddings. 2) GSim+ time rises in proportion to the size $|\mathcal{G}_A|$, which requires only 1.78s on small HP and 1794s on large IT. This is compliant with the complexity analysis in Theorem 4.1. 3) Across all datasets, SS-BC* is consistently 3–67x slower than GSim+ due to SS-BC*'s repeated execution of single-pair queries for $|Q_A| \times |Q_B|$ query pairs, resulting in many duplicate computations. 4) RSim survives on small HP only, as it requires cubic time for the recursive computation of maximal neighbor matching for every pair of nodes. 5) NED is only viable on small HP, and is 10,430x slower than GSim+, due to the unbounded size of NED's k-adjacent trees.

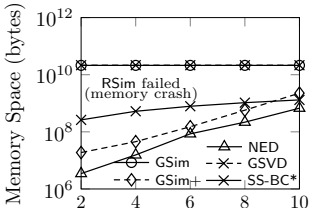
Varying k. To show the impact of k on GSim+ time, we fix query size $(|Q_A|, |Q_B|)$ and vary k from 2 to 10 on each dataset. in Figure 3. It is discernible that 1) on each dataset, the time of GSim+ grows mildly as k rises, as expected. 2) NED performs well only with small k on HP, but it fails to yield results within one day for larger graphs (WT and IT). This is because the number of nodes per level in the unbounded k-adjacent trees experiences an exponential increase *w.r.t.* k. 3) RSim struggles with all these datasets since it cannot memoise all-pairs similarities at each iteration, even for partial pair queries. 4) SS-BC* works well on larger WT and IT but consistently lags behind GSim+ in terms of speed across all iterations, indicating its inefficiency in repeatedly executing single-pair queries for handling partial-pair queries. 5) When GSim and GSVD survive, GSim+ is consistently faster by at least 2–3 orders of magnitude. 6) Even when GSim and GSVD experience memory crash on WT, GSim+ still scales well due to the effective use of the progressive low-rank embedding of S. It is important to note that, in practice, when k is small (*e.g.* around 10), decent accuracy can be achieved on real datasets. For instance, when $k = 12$, the error of GSim+ on the HP dataset is already below 0.01. As k increases, accuracy improves, but it comes at the expense of increased time and memory requirements. Thus, choosing k involves a trade-off between speed and accuracy. More importantly, even when k becomes larger, the time and space required by GSim+ never surpass those of GSim. This is because, when the low dimensionality (2^k) exceeds the number of graph nodes (n_A or n_B), GSim similarity computation reverts to the original space dimensionality of $\min\{n_A, n_B\}$. In such cases, after the iterations where $2^k \geq \min\{n_A, n_B\}$, GSim+ reduces to the traditional GSim without dimensionality reduction. Consequently, to perform the total number of iterations, the time and space required by GSim+ remain always no greater than those of GSim.

Effect of $|\mathcal{V}_B|$ on Time. Figure 4 shows how the GSim+ time is impacted by $|\mathcal{G}_B|$. 1) GSim+ scales well *w.r.t.* $|\mathcal{V}_B|$ on all datasets, and runs 2-3 orders of magnitude faster than GSim as $|\mathcal{V}_B|$ rises, showing the scalability of our low-embeddings on large graphs.

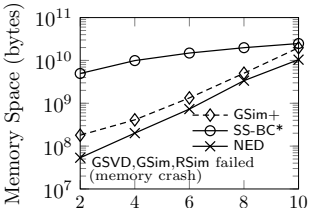


(a) Vary $\mathcal{G}_A$ on Real Datasets

Figure 6: Memory Space

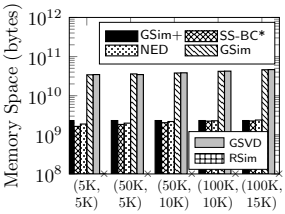


(a) Vary k on EE



(b) Vary k on WT

Figure 7: Effect of k on Memory Space over EE and WT



(a) Vary $(|Q_A|, |Q_B|)$ on EE

Figure 8: Effect of $|Q_A|, |Q_B|$

2) As $\mathcal{G}_B$ grows, GSim and GSVD are more sensitive to $|\mathcal{V}_B|$ since GSim needs to assess the entire S_k per iteration, whereas GSim+ uses the low-embedding representation. 3) SS-BC*, being a local single-pair search algorithm, shows little sensitivity to $|\mathcal{G}_B|$, as its runtime is mainly influenced by the size of query sets and the k-hop neighboring structure of each pair. 4) NED remains unresponsive due to its costly maximum matching in unbounded k-adjacent trees for each query. 5) RSim crashes on EE and WT due to its substantial space requirements for storing all-pair similarities.

Effect of $(|Q_A|, |Q_B|)$ on Time. Figure 5 depicts how changes in query size ($|Q_A|$ and $|Q_B|$) affect the time of GSim+ and its rivals. 1) On each dataset, GSim+ shows impressive scalability, with only a mild increase in time as $|Q_A|$ and $|Q_B|$ grows. 2) As Q_A and Q_B increase in size, SS-BC* is always slower than GSim+ and becomes more sensitive to the size of query sets. This aligns with its time complexity since SS-BC* addresses partial-pair queries by executing a single query multiple times. 3) NED remains unresponsive for both graphs due to the exponential growth of the k-adjacent trees.

5.2.2 Memory Efficiency. We next compare GSim+ memory with its rivals and show how it is affected by various parameters.

Varying $\mathcal{G}_A$ on Real Datasets. Figure 6 compares memory between GSim+ and its competitors. 1) On all datasets, GSim+ entails 5–15 times less memory than GSim and GSVD, and scales well on large graphs. GSim and GSVD crash on large graphs due to their need for huge memory to maintain a large dense S. The memory of GSim+ is comparable with SS-BC* and NED and is slightly higher. This is because SS-BC* and NED are single-pair search algorithms that will release memory after calculation of each pair of similarity. 2) GSim+ memory rises linearly with the size of $\mathcal{G}_A$, under the same iterations. This agrees with our space complexity in Section 4.

Varying k. Figure 7 shows the impact of k on GSim+ memory. 1) As k grows, GSim+ always consumes 1–3 orders of magnitude less memory than GSim and GSVD on EE. Moreover, GSim+ scales well on large WT, unlike GSim and GSVD suffering from memory explosion. This highlights the efficacy of GSim+ that incrementally generates the low-rank embedding for S. 2) NED memory is on par with GSim+ across all iterations because NED promptly frees up memory from the similarity results of the current node pair to evaluate the similarity of the node pair in the query set. 3) SS-BC* consumes additional memory compared to GSim+ because of its need for space to accommodate the index.

Effect of $(|Q_A|, |Q_B|)$ on Space. Figure 8 shows the impact of $(|Q_A|, |Q_B|)$ on the memory of GSim+ and its rivals. Due to similar trends, we only report results on EE for space interest. We see that, 1) as $|Q_A|$ and $|Q_B|$ rise, GSim+ shows superior scalability to GSim and GSVD. This is understandable given that GSim and GSVD both require huge memory to keep all-pairs similarities, whereas GSim+ only needs to store low-embeddings of S. 2) RSim encounters memory issues when attempting to materialise all-pairs similarities, leading to its failure. 3) GSim+, SS-BC*, and NED have comparable memory requirements. They are approximately 1.5 orders of magnitude lower than GSim and GSVD, being less sensitive to query size, which is consistent with the space complexity.

5.2.3 Accuracy of Approximation. Lastly, we evaluate the accuracy of GSim+ on real datasets and compare it with GSim+ (with low-rank $r \in \{5, 10, 50\}$). Below, we only show results on HP due to its similarity to other datasets. The accuracy is measured by the error $\|S_k - S\|_F$, where S_k is the similarity matrix at k-th iteration generated by each algorithm, and S is the exact solution performed by GSim for 100 iterations, ensuring that the similarity values are accurate up to 6 decimal places, achieving 'float' type precision.

k	GSim+ / GSim	GSVD		
		($r = 5$)	($r = 10$)	($r = 50$)
4	1.89755×10^{-2}	2.13657×10^{-2}	2.13632×10^{-2}	2.13628×10^{-2}
8	1.03769×10^{-2}	1.24451×10^{-2}	1.24352×10^{-2}	1.24350×10^{-2}
12	6.17055×10^{-3}	8.11206×10^{-3}	8.09702×10^{-3}	8.09688×10^{-3}
16	3.79280×10^{-3}	5.74970×10^{-3}	5.73933×10^{-3}	5.73927×10^{-2}
20	2.34014×10^{-3}	4.41413×10^{-3}	4.40861×10^{-3}	4.40859×10^{-2}

Here are the key findings: 1) For each iteration k, the errors of GSVD slightly decrease as low-rank r grows. However, even with this decrease, the errors of GSVD remain consistently larger than those of GSim+, regardless of the chosen low-rank r. This shows that our low-embeddings effectively avoid the issues of overfitting and underestimation in similarity approximation. 2) The errors of GSim+ and GSim are identical at each iteration. This indicates that our low embedding technique does not compromise the accuracy of GSim, highlighting the correctness of Theorem 3.1. 3) The error of GSim+ approaches zero as k rises, and its convergence rate outpaces that of GSVD. This aligns with our error analysis in Theorem 4.2.

6 CONCLUSIONS

This paper studies the efficient computation of GSim similarity between a collection of nodes across two graphs at a large scale with guaranteed accuracy. Firstly, we propose a novel algorithm namely GSim+, which employs an adaptive low-dimensional embedding technique for iteratively evaluating the similarity matrix, greatly accelerating the computation of GSim. Secondly, we provide a theoretical error bound on the iterative computation of GSim+, and analyse its computational complexity. Finally, our experiments on real datasets demonstrate that GSim+ surpasses state-of-the-art competitors, achieving 1–4 orders of magnitude speedup and superior scalability on billion-edge graphs.

Acknowledgments. This work is supported by the National Natural Science Foundation of China under Grant No. 61972203.

REFERENCES

[1] [n.d.]. DBLP Data. https://dblp.uni-trier.de/xml/.

[2] [n.d.]. LAW Datasets. https://sparse.tamu.edu/LAW.

[3] [n.d.]. SNAP Datasets. https://snap.stanford.edu/data/.

[4] Vincent D Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. 2004. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM review* 46, 4 (2004), 647–666.

[5] Elizabeth E Bruch and MEJ Newman. 2019. Structure of online dating markets in US cities. *Sociological Science* 6 (2019), 219–234.

[6] Thomas P Cason, Pierre-Antoine Absil, and Paul Van Dooren. 2013. Iterative methods for low rank approximation of graph similarity matrices. *Linear Algebra Appl.* 438, 4 (2013), 1863–1882.

[7] Xiaoshuang Chen, Longbin Lai, Lu Qin, and Xuemin Lin. 2021. Efficient structural node similarity computation on billion-scale graphs. *The VLDB Journal* 30 (2021), 471–493.

[8] Catherine Fraikin, Yurii Nesterov, and Paul Van Dooren. 2008. Optimizing the coupling between two isometric projections of matrices. *SIAM journal on matrix analysis and applications* 30, 1 (2008), 324–345.

[9] Aron Henriksson, Hans Moen, Maria Skeppstedt, Vidas Daudaravičius, and Martin Duneld. 2014. Synonym extraction and abbreviation expansion with ensembles of semantic spaces. *Journal of biomedical semantics* 5, 1 (2014), 1–25.

[10] 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.

[11] Ruoming Jin, Victor E Lee, and Hui Hong. 2011. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 922–930.

[12] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

[13] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. 2006. Vertex similarity in networks. *Physical Review E* 73, 2 (2006), 026120.

[14] 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.

[15] Tova Milo, Amit Somech, and Brit Youngmann. 2019. Boosting SimRank with semantics. In *Proceedings of the 22th International Conference on Extending Database Technology*. 1–12.

[16] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications* 1 (2010), 19–30.

[17] Sascha Rothe and Hinrich Schütze. 2014. CoSimRank: A flexible & efficient graph-theoretic similarity measure. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1392–1402.

[18] Weiren Yu, Julie A. McCann, Chengyuan Zhang, and Hakan Ferhatosmanoglu. 2022. Scaling High-Quality Pairwise Link-Based Similarity Retrieval on Billion-Edge Graphs. *ACM Trans. Inf. Syst.* 40, 4 (2022), 78:1–78:45. https://doi.org/10.1145/3495209

[19] Weiren Yu and Fan Wang. 2018. Fast Exact CoSimRank Search on Evolving and Static Graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. ACM, 599–608. https://doi.org/10.1145/3178876.3186126

[20] Weiren Yu, Jian Yang, Maoyin Zhang, and Di Wu. 2022. CoSimHeat: An Effective Heat Kernel Similarity Measure Based on Billion-Scale Network Topology. In *Proceedings of the ACM Web Conference 2022*. 234–245.

[21] Laura A Zager and George C Verghese. 2008. Graph similarity scoring and matching. *Applied mathematics letters* 21, 1 (2008), 86–94.

[22] Haohan Zhu, Xianrui Meng, and George Kollios. 2016. NED: an inter-graph node metric based on edit distance. *arXiv preprint arXiv:1602.02358* (2016).