CSR+: A Scalable Efficient CoSimRank Search Algorithm with Multi-Source Queries on Massive Graphs

Maoyin Zhang
Nanjing University of Sci. & Tech.
Jiangsu, Nanjing, China
maoyinzhang@njust.edu.cn

Weiren Yu[*]
Warwick University
Coventry, CV4 7AL, UK
weiren.yu@warwick.ac.uk

ABSTRACT

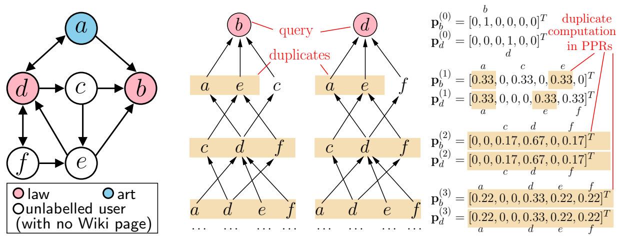
CoSimRank is a compelling model that iteratively follows the SimRank-like notion that "two items (nodes) are deemed similar if their in-neighbours are similar". However, the existing CoSimRank algorithm [6] is only efficient in single-source search, and is rather sluggish for multi-source queries due to repeated calculations across queries. Another algorithm [4] involves cost-inhibitive tensor products in preprocessing, rendering it unscalable on big graphs. In this paper, we propose CSR+, a fast efficient multi-source CoSimRank algorithm that scales well on billion-edge graphs. First, we analyse the pros and cons of [4] and propose a four-stage optimisation scheme to eliminate costly graph tensor products and numerous repetitive similarity calculations while addressing each of the deficiencies in [4]. Next, we present an efficient algorithm, CSR+, for multi-source CoSimRank search, which significantly reduces the computation time from $O(r^4n^2)$ to $O(r(m+n(r+|Q|)))$ and memory from $O(r^2n^2)$ to $O(rn)$ on a graph of n nodes and m edges, without compromising the accuracy of [4], where r ($\ll n$) is the low rank of singular value decomposition. Experiments on various real datasets validate that CSR+ is 1—4 orders of magnitude faster than its rivals for multi-source queries while scaling on large graphs.

1 INTRODUCTION

Evaluating similarity between nodes is an essential operation on graphs, used in a wide spectrum of real applications, *e.g.* synonym expansion [10], knowledge graphs [16], link prediction [7], and so forth. Many graph-theoretic similarity measures have been proposed, including SimRank [1], SimFusion [8], CoSimRank [6], SemSim [5], and CoSimHeat [15]. Of these, CoSimRank, conceived by Rothe and Schütze [6], shines out as particularly appealing due to its simple SimRank-like intuition that "two items (nodes) are deemed similar if their in-neighbours are similar". However, CoSimRank similarity between each node and itself is not necessarily 1. This base case differs from the well-known SimRank definition [1], where the similarity of every node to itself is constantly 1. Due to this nuance, CoSimRank and SimRank interpret random surfers differently: CoSimRank similarity for nodes a and b reflects all the meeting times of two random surfers starting from nodes a and b, while SimRank reflects only their first meeting time. Therefore, CoSimRank contains richer link information than SimRank.

In this study, we consider the problem of fast CoSimRank search with *multi-source queries.* Given a graph $\mathcal{G}$ and a set of queries Q, we want to retrieve CoSimRank similarities, $[\mathrm{S}]_{*,Q} := \{[\mathrm{S}]_{x,q}\}_{x\in\mathcal{G},q\in Q}$, between each node in $\mathcal{G}$ and each query q in Q. Many real-world applications benefit from multi-source similarity search, *e.g.* document classification, and social community identification.

[*]Weiren Yu is the corresponding author.


a
b
c
d
e
f
law
art
unlabelled user (with no Wiki page)
query
duplicates
duplicate computation in PPRs
$\mathbf{p}_b^{(0)} = [0,1,0,0,0,0]^T$
$\mathbf{p}_d^{(0)} = [0,0,0,1,0,0]^T$
$\mathbf{p}_b^{(1)} = [0.33,0,0.33,0,0.33,0]^T$
$\mathbf{p}_d^{(1)} = [0.33,0,0,0,0.33,0.33]^T$
$\mathbf{p}_b^{(2)} = [0,0,0.17,0.67,0,0.17]^T$
$\mathbf{p}_d^{(2)} = [0,0,0.17,0.67,0,0.17]^T$
$\mathbf{p}_b^{(3)} = [0.22,0,0,0.33,0.22,0.22]^T$
$\mathbf{p}_d^{(3)} = [0.22,0,0,0.33,0.22,0.22]^T$

(a) Wiki Talk Graph $\mathcal{G}$ (b) 3-hop In-linked Propagation from Each Individual Query

Figure 1: Duplicate Computations in CoSimRank Search $[\mathrm{S}]_{*,Q}$ with Multi-Source Queries $Q = \{b, d\}$ on Graph $\mathcal{G}$

Application (Wikipedians Categorisation). Figure 1(a) depicts a tiny Wikipedia Talk graph, where a node is a Wikipedia user, and an edge $x \longrightarrow y$ exists if user x edited user y's talk page. Some users (*e.g.* $\{a, b, d\}$), who have added their user page to the "Wikipedian-by-interest" category, are labelled with an area of interest (*e.g.* "art" for user a and "law" for b and d). To retrieve users in $\mathcal{G}$ relevant to the area of "law", similarities of all nodes in $\mathcal{G}$ are evaluated *w.r.t.* a set of queries $Q = \{b, d\}$ labelled with "law". Similarity values $[\mathrm{S}]_{*,Q}$ can aid in automatic Wikipedians categorisation based on interests, using the link structure of Wikipedia Talk. □

However, existing work on CoSimRank computation suffers from two limitations: Firstly, the previous algorithm [6] is efficient only in single-source search, and is too sluggish for multi-source queries due to repeated calculations across queries, as shown in Example 1.1.

Example 1.1. In Figure 1(a), given a multi-source query $Q = \{b, d\}$, we aim to retrieve the CoSimRank similarities $[\mathrm{S}]_{*,Q}$. The existing algorithm [6] splits Q into two single-source individual queries and evaluates $[\mathrm{S}]_{*,b}$ and $[\mathrm{S}]_{*,d}$ independently, leading to many duplicate calculations. Precisely, for each query b (*resp.* d), first, the Personalised PageRank (PPR) vector *w.r.t.* seed node b (*resp.* d) at each iteration k is computed to get the k-hop in-neighbours of b (*resp.* d). As shown in Figure 1(b), the 1-hop in-neighbour sets of b and d share two nodes $\{a, e\}$. Since c and f have the same in-neighbour set $\{d\}$, b and d have the same 2-hop in-neighbour sets, leading to identical PPR vectors ($\mathbf{p}_b^{(k)} = \mathbf{p}_d^{(k)}$) for every subsequent iteration $k = 2, 3, \cdots$. These identical PPRs (in orange) indicate many duplicate computations. □

Secondly, another optimisation algorithm proposed by Li *et al.* [4] divides similarity calculation into two stages: precomputation and online query. However, the low-rank decomposition in precomputation entails a cost-inhibitive graph tensor product. Worse, there are many superfluous memorised results from precomputation which are futile for online query, as will be detailed in Sections 3.1–3.2. It is worth mentioning that, although the initial

Algorithm	Time	Memory	Error
CSR+ (this work)	$O(r(m+n(r+\|Q\|)))$	$O(rn)$	the same low rank-r error
NI-Sim [4]	$O(r^4n^2+r^4n\|Q\|)$	$O(r^2n^2)$	
CoSimRank [6]	$O(n^2\log(1/\epsilon)\|Q\|)$	$O(n^2)$	ϵ
CoSimMate [11]	$O(n^3\log_2(\log(1/\epsilon)))$	$O(n^2)$	ϵ
PR-CoSim [9]	$O(n^2\log(n)/\epsilon^2\log(1/\epsilon))$	$O(n^2)$	ϵ
F-CoSim [14]	$O(n^2+\log(1/\epsilon)n(m-n)\|Q\|)$	$O(n^2)$	ϵ

Table 1: Comparison of Various CoSimRank Algorithms for Multi-Source Search $[\mathbf{S}]_{*,Q}$ On Query Set Q ($n\times|Q|$ Pairs)

intention of [4] was to speed up SimRank search, the approach in that work actually provides an approximation of SimRank similarity using a linear matrix equation. This equation [4, Eq.(2)] lacked a specific name for the similarity measure, but later research [13] proved it to be an exact scaled version of the CoSimRank equation. Consequently, the methodologies presented by [4] essentially represent optimization techniques for the CoSimRank measure. Thus, there is a pressing need to devise a novel efficient CoSimRank algorithm.

Contributions. Our main contributions are listed as follows:

- We first discuss the pros and cons of Li *et al.*'s work [4] (§ 3.1), and present optimisation techniques to eliminate its costly tensor products and many repeated calculations during similarity computation while tackling each of the blemishes in [4] (§ 3.2).
- We propose an efficient algorithm, CSR+, for multi-source CoSimRank search with theoretical guarantees, which greatly reduces the time [4] from $O(r^4n^2)$ to $O(r(m+n(r+|Q|)))$ and memory from $O(n^2r^2)$ to $O(nr)$ on a graph of n nodes and m edges (§ 3.3).
- Using various real datasets, we empirically validate that CSR+ consistently outperforms the state-of-the-art rivals by $1-4$ orders of magnitude in both time and space for multi-source queries while scaling effectively on billion-edge graphs (§ 4).

Related Work. Recent years have witnessed an upsurge of interest in the efficient search for CoSimRank since its conception by Rothe and Schütze [6]. They also suggested two efficient algorithms for single-pair and single-source CoSimRank search, respectively. However, when broadened to multi-source queries, such approaches become exceedingly slow due to a number of repeated calculations across distinct queries. Since then, a variety of methods for accelerating the CoSimRank computation [4, 9, 11, 14] have been presented. Their time and space complexity for multi-source CoSimRank search has been summarised in Table 1. CoSimmate [11] adopts a repeated squaring approach, which can exponentially cut down the number of iterations to guarantee the same accuracy, but requires quadratic space to memoise the squared intermediate results. As a result, this approach is unsuitable for direct use in high-dimensional space. Renchi Yang [9] presented RP-CoSim, a randomised technique for estimating all-pairs CoSimRank similarities by Gaussian sampling. Nevertheless, its $O(n^2)$ memory seriously limits its scalability on large graphs. Recently, Yu and Fan [14] proposed a dynamic single-source CoSimRank method for evolving networks. This strategy, however, is less efficient when employed for multi-source search on static graphs. Recently, there is also another method [4] using low-rank SVD to speed up SimRank computation, but [4] worked on the linear approximation of the SimRank equation, which essentially generates the scaled CoSimRank scores. Please refer to Section 2 for more details. Moreover, there exist rather costly graph tensor products in precomputation, making [4] unscalable on any large graphs.

Symbol	Description	Symbol	Description
$\mathcal{G}(\mathcal{V},\mathcal{E})$	graph ($\mathcal{V}/\mathcal{E}$: node/edge set)	Q	a set of queries
$[\mathbf{X}]_{i,*}$	i-th row of matrix $\mathbf{X}$	$\mathbf{Q}$	transition matrix
$[\mathbf{X}]_{*,j}$	j-th column of matrix $\mathbf{X}$	$\mathbf{I}_n$	$n\times n$ identity matrix
$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$	$\mathbf{S}$	similarity matrix
n, m	number of nodes/edges in G	ϵ	desired accuracy
r	target low rank of SVD	c	damping factor

Table 2: Description of Main Symbols

2 PRELIMINARIES

We formally revisit the origins of CoSimRank [6]. The key symbols and definitions used in this paper are laid out in Table 2.

CoSimRank Formulation. Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$, let $\mathbf{Q}$ be the column-normalised adjacency matrix. The CoSimRank matrix $\mathbf{S}$ is defined as

$$\mathbf{S}=c\mathbf{Q}^T\mathbf{S}\mathbf{Q}+\mathbf{I}_n \tag{1}$$

where $c\in(0,1)$ is a user-specified damping factor, typically set to 0.6 or 0.8 [6]. $(*)^T$ is matrix transpose. $\mathbf{I}_n$ is a $n\times n$ identity matrix.

In comparison, CoSimRank Eq.(1) is similar to the following Jeh and Widom's well-known SimRank equation [1]:

$$\mathbf{S}=\max\{c\mathbf{Q}^T\mathbf{S}\mathbf{Q},\ \mathbf{I}_n\} \tag{2}$$

Contrary to SimRank with the entry-wise max operator "$\max\{*,\mathbf{I}_n\}$" which sets each element on the diagonal of $\mathbf{S}$ to 1, CoSimRank Eq.(1) with "$+\,\mathbf{I}_n$" ensures that each similarity $[\mathbf{S}]_{a,a}$ not only recursively considers the in-neighbouring structure of node a but also exceeds the similarity value $[\mathbf{S}]_{a,x}$ between a and any other node x in $\mathcal{G}$.

CoSimRank Computation. Rothe and Schütze [6] computes each pair of CoSimRank similarity as follows:

$$[\mathbf{S}]_{a,b}=\sum_{k=0}^{\infty}c^k(\mathbf{p}_a^{(k)})^T\mathbf{p}_b^{(k)} \tag{3}$$

where $\mathbf{p}_a^{(k)}$ (*resp.* $\mathbf{p}_b^{(k)}$) is the k-th iterative Personalised PageRank vector *w.r.t.* seed node a (*resp.* b), which is iteratively computed as

$$\mathbf{p}_a^{(k+1)}=\mathbf{Q}\mathbf{p}_a^{(k)}\ (k=0,1,\cdots)\quad\text{with}\quad\mathbf{p}_a^{(0)}=[\mathbf{I}_n]_{*,a}$$

For single-pair or single-source similarity search, the method of Eq.(3) is scalable and efficient. However, when extended to multi-source queries, this method becomes rather sluggish.

Li *et al.*'s [4] Relation to CoSimRank. To support multi-source or all-pairs similarity search, Li *et al.* [4] suggested a non-iterative method to efficiently solve the matrix $\mathbf{S}'$ to the following equation:

$$\mathbf{S}'=c\mathbf{Q}^T\mathbf{S}'\mathbf{Q}+(1-c)\mathbf{I}_n \tag{4}$$

and considered the solution $\mathbf{S}'$ to Eq. (4) as a linear approximation of the SimRank matrix $\mathbf{S}$ to Eq.(2). The recent work [13] has proved that the solution $\mathbf{S}'$ in Eq.(4) is different from the SimRank solution $\mathbf{S}$ in Eq.(2). Moreover, it can be easily verified that $\mathbf{S}'$ in Eq.(4) is exactly the scaled version of the CoSimRank solution $\mathbf{S}$ to Eq.(1). That is, $\mathbf{S}'$ and $\mathbf{S}$ satisfy the relationship: $\mathbf{S}'=(1-c)\mathbf{S}$. Therefore, the computational methods of [4] to solve Eq.(4) for $\mathbf{S}'$ can be regarded as an algorithm designed to compute the scaled CoSimRank similarity, rather than the SimRank values. This is the reason why the work [4] is perceived more closely to CoSimRank than SimRank. As a result, the non-iterative optimisation techniques of Li *et al.* [4] for solving Eq.(4) can be directly applied to CoSimRank computation. Specifically, Li *et al.* [4] first introduced two matrix operators:

Definition 2.1. vec(X) vectorises a $p \times q$ matrix $\mathbf{X}$ into the $pq \times 1$ column vector by stacking the columns of $\mathbf{X}$ on top of one another:

$$\text{vec}(\mathbf{X}) = [x_{11}, \cdots, x_{q1}, x_{12}, \cdots, x_{q2}, \cdots, x_{1p}, \cdots, x_{qp}]^T \quad \square$$

Definition 2.2. The tensor product $\mathbf{X} \otimes \mathbf{Y}$ of a $p \times q$ matrix $\mathbf{X}$ and a $r \times s$ matrix $\mathbf{Y}$ is the $pr \times qs$ matrix, defined as

$$\mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{11}\mathbf{Y} & \ldots & x_{1q}\mathbf{Y} \\ \vdots & \ddots & \vdots \\ x_{p1}\mathbf{Y} & \ldots & x_{pq}\mathbf{Y} \end{bmatrix} \quad \square$$

Employing these two operators, Li *et al.* [4] derives the following closed-form for the similarity matrix $\mathbf{S}$ to Eq.(1):

$$\text{vec}(\mathbf{S}) = \left(\mathbf{I}_{n^2} - c(\mathbf{Q} \otimes \mathbf{Q})^T\right)^{-1} \text{vec}(\mathbf{I}_n) \tag{5}$$

To speed up the computation of matrix inverse in Eq.(5), Li *et al.* [4] next applied the SVD ($\mathbf{Q} = \mathbf{U}\Sigma\mathbf{V}^T$) to the tensor product $\mathbf{Q} \otimes \mathbf{Q}$ in conjunction with the Sherman-Morrison formula, yielding the following expression to compute similarity matrix $\mathbf{S}$ in Eq.(1):

$$\text{vec}(\mathbf{S}) = \text{vec}(\mathbf{I}_n) + c(\mathbf{U} \otimes \mathbf{U})\Lambda(\mathbf{V} \otimes \mathbf{V})^T \text{vec}(\mathbf{I}_n) \tag{6a}$$

$$\text{with } \Lambda = \left((\Sigma \otimes \Sigma)^{-1} - c(\mathbf{V} \otimes \mathbf{V})^T(\mathbf{U} \otimes \mathbf{U})\right)^{-1} \tag{6b}$$

The pros and cons of this method will be discussed in Section 3.1.

3 OPTIMISING COSIMRANK COMPUTATION

3.1 Pros and Cons of Li *et al.*'s Approach

A key advantage of Li *et al.*'s method [4] is that the computation of similarity is divided into two phases: 1) In the preprocessing phase, the matrix Λ is computed only once by Eq.(6b). 2) In the query phase, Λ is reused and shared multiple times to compute the multi-source similarity $[\mathbf{S}]_{*,Q}$ over a set of queries Q via Eq.(6a). However, there are several limitations associated with Li *et al.*'s method [4] that impede its scalability and speed:

- A costly tensor product $(\mathbf{V} \otimes \mathbf{V})^T(\mathbf{U} \otimes \mathbf{U})$ in Eq.(6b), which requires $O(r^4n^2)$ time and $O(r^2n^2)$ memory.
- A redundant $O(r^2n^2)$-time tensor product $(\mathbf{V} \otimes \mathbf{V})^T$ in Eq.(6a), whose removal has no effect on the outcome.
- Many elements of Λ precomputed by Eq.(6b) are not used at all for subsequent similarity computation via Eq.(6a).
- An expensive $O(r^2n^2)$-time tensor product $(\mathbf{U}\otimes\mathbf{U})$ in Eq.(6a), which can be eliminated without affecting the result.

Consequently, this method does not scale well on large graphs since r^4 may be much larger than n even for the low rank $r \ll n$.

3.2 Our Optimisation Techniques

To greatly reduce the computational cost of Li *et al.*'s method [4], we next propose an efficient four-stage optimisation scheme for addressing each of the deficiencies outlined in Section 3.1.

1) Speeding up Computation of $(\mathbf{V} \otimes \mathbf{V})^T(\mathbf{U} \otimes \mathbf{U})$ in Eq.(6b). Our first optimisation method rests on the following observation:

THEOREM 3.1. *The most expensive part $(\mathbf{V} \otimes \mathbf{V})^T(\mathbf{U} \otimes \mathbf{U})$ in Eq.(6b) can be computed efficiently as*

$$(\mathbf{V} \otimes \mathbf{V})^T(\mathbf{U} \otimes \mathbf{U}) = \Theta \otimes \Theta \quad \textit{with} \quad \Theta = \mathbf{V}^T\mathbf{U} \tag{7}$$

which substantially reduces the computational time from $O(r^4n^2)$ to $O(r^2n + r^4)$, and the memory usage from $O(r^2n^2)$ to $O(rn + r^4)$.

PROOF. Eq.(7) is based on two properties of the tensor product:

(1) Transpositions are distributive over the tensor product, *i.e.*, $(\mathbf{V} \otimes \mathbf{V})^T = \mathbf{V}^T \otimes \mathbf{V}^T$.
(2) Mixed product property, *i.e.*, if $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are matrices of such size that one can form the matrix products $\mathbf{AC}$ and $\mathbf{BD}$, then

$$(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = (\mathbf{AC})\otimes(\mathbf{BD}) \text{ with } \mathbf{A} = \mathbf{B} = \mathbf{V}^T \text{ and } \mathbf{C} = \mathbf{D} = \mathbf{U}.$$

Combining these two properties, Eq.(7) follows immediately. $\square$

Note that, in Eq.(7), computing $\Theta = \mathbf{V}^T\mathbf{U}$ only requires $O(r^2n)$ time and $O(rn)$ memory. Since Θ is of size $r \times r$, the computation of $\Theta \otimes \Theta$ entails $O(r^4)$ time and $O(r^4)$ memory. Moreover, Θ only needs to be computed once and is reused twice to obtain $\Theta \otimes \Theta$. As a result, the overall cost of Eq.(7) requires $O(r^2n + r^4)$ time and $O(rn + r^4)$ memory, which is a significant improvement over the $O(r^4n^2)$ time and $O(r^2n^2)$ memory of Li *et al.*'s approach [4].

2) Removing Unnecessary Tensor Product $(\mathbf{V}\otimes\mathbf{V})^T$ in Eq.(6a). Our second observation is based on the following theorem:

THEOREM 3.2. *For the query phase, computing the tensor product $(\mathbf{V} \otimes \mathbf{V})^T$ in Eq.(6a) is redundant, i.e., Eq.(6a) can be simplified as*

$$vec(\mathbf{S}) = vec(\mathbf{I}_n) + c(\mathbf{U} \otimes \mathbf{U})(\Lambda vec(\mathbf{I}_r)) \tag{8}$$

PROOF. After the SVD decomposition $\mathbf{Q} = \mathbf{U}\Sigma\mathbf{V}^T$ is applied, the $n \times r$ matrix $\mathbf{V}$ is column-orthonormal, which satisfies $\mathbf{I}_r = \mathbf{V}^T\mathbf{V}$. We take vec($*$) on both sides and apply the tensor property to get

$$\text{vec}(\mathbf{I}_r) = \text{vec}(\mathbf{V}^T\mathbf{V}) = (\mathbf{V}^T \otimes \mathbf{V}^T)\text{vec}(\mathbf{I}_n) = (\mathbf{V} \otimes \mathbf{V})^T\text{vec}(\mathbf{I}_n)$$

Replacing $(\mathbf{V} \otimes \mathbf{V})^T\text{vec}(\mathbf{I}_n)$ in Eq.(6a) with $\text{vec}(\mathbf{I}_r)$ yields Eq.(8). $\square$

Theorem 3.2 not only accelerates the computation of Eq.(6a) by saving the $O(r^2n^2)$ time and memory of the tensor product $(\mathbf{V} \otimes \mathbf{V})$ without compromising any accuracy, but also reveals much redundancy in Eq.(8), which directs our next optimisation technique.

3) Eliminating Redundancy of $(\Lambda\text{vec}(\mathbf{I}_r))$ in Eq.(8). Note that Λ is the $r^2 \times r^2$ matrix precomputed by Eq.(6b), and $\text{vec}(\mathbf{I}_r)$ is a very sparse vector with only r elements 1s and the remaining $r \times (r-1)$ elements 0s. As a result, the computation of $(\Lambda\text{vec}(\mathbf{I}_r))$ is essentially to extract only r columns (with their column indices associated with the indices of the 1s in $\text{vec}(\mathbf{I}_r)$) from the r^2 columns of matrix Λ. This implies that all the remaining $r \times (r-1)$ columns of Λ is useless for $(\Lambda\text{vec}(\mathbf{I}_r))$ computation in Eq.(8). Consequently, there is no need to precompute all $r^2 \times r^2$ elements of Λ via Eq.(6b).

Nonetheless, to eliminate unnecessary computations in $(\Lambda\text{vec}(\mathbf{I}_r))$, it seems challenging to directly resort to the expression of Λ in Eq.(6b) for speedup. Thus, we begin by representing Λ as another expression, which serves as the basis for $(\Lambda\text{vec}(\mathbf{I}_r))$ optimisation.

THEOREM 3.3. *The matrix Λ in Eq.(6b) can be expressed as*

$$\Lambda = (\Sigma \otimes \Sigma)\left(\mathbf{I}_{r^2} - c(\mathbf{H} \otimes \mathbf{H})\right)^{-1} \quad \textit{with} \quad \mathbf{H} = \mathbf{V}^T\mathbf{U}\Sigma \tag{9}$$

PROOF. In virtue of Eq.(6b) and Theorem 3.1, it follows that

$$(\Sigma \otimes \Sigma)^{-1}\Lambda = \underbrace{(\Sigma \otimes \Sigma)}_{=\mathbf{X}}{}^{-1}\Big(\underbrace{(\Sigma \otimes \Sigma)^{-1} - c(\mathbf{V}^T\mathbf{U}) \otimes (\mathbf{V}^T\mathbf{U})}_{=\mathbf{Y}}\Big)^{-1}$$

Applying the inverse property $\mathbf{X}^{-1}\mathbf{Y}^{-1} = (\mathbf{YX})^{-1}$, we have

$$\begin{aligned}(\Sigma \otimes \Sigma)^{-1}\Lambda &= \left(\mathbf{I}_{r^2} - c\left((\mathbf{V}^T\mathbf{U}) \otimes (\mathbf{V}^T\mathbf{U})\right)(\Sigma \otimes \Sigma)\right)^{-1} \\ &= \left(\mathbf{I}_{r^2} - c(\mathbf{H} \otimes \mathbf{H})\right)^{-1} \quad \text{with} \quad \mathbf{H} = \mathbf{V}^T\mathbf{U}\Sigma\end{aligned}$$

Left-multiplying both sides by $(\Sigma \otimes \Sigma)$ produces Eq.(9). □

Theorem 3.3 is introduced to optimise the computation of $(\Lambda \text{vec}(\mathbf{I}_r))$ in Eq.(8), which is based on the following theorem.

THEOREM 3.4. *The term $(\Lambda vec(\mathbf{I}_r))$ in Eq.(8) can be computed as*

$$\Lambda vec(\mathbf{I}_r) = vec(\Sigma \mathbf{P} \Sigma) \tag{10}$$

where $\mathbf{P}$ *is the solution to the following equation (in $r \times r$ subspace):*

$$\mathbf{P} = c\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{I}_r \quad \textit{with} \quad \mathbf{H} = \mathbf{V}^T\mathbf{U}\Sigma \tag{11}$$

PROOF. Plugging Λ in Eq.(9) into $\Lambda \text{vec}(\mathbf{I}_r)$, we obtain

$$\Lambda \text{vec}(\mathbf{I}_r) = (\Sigma \otimes \Sigma)\big(\mathbf{I}_{r^2} - c(\mathbf{H} \otimes \mathbf{H})\big)^{-1}\text{vec}(\mathbf{I}_r) = (\Sigma \otimes \Sigma)\text{vec}(\mathbf{P})$$

where $\text{vec}(\mathbf{P})$ satisfies $\big(\mathbf{I}_{r^2} - c(\mathbf{H} \otimes \mathbf{H})\big)\text{vec}(\mathbf{P}) = \text{vec}(\mathbf{I}_r)$, *i.e.*,

$$\text{vec}(\mathbf{P}) = c(\mathbf{H} \otimes \mathbf{H})\text{vec}(\mathbf{P}) + \text{vec}(\mathbf{I}_r) = \text{vec}(c\mathbf{H}\mathbf{P}\mathbf{H}^T) + \text{vec}(\mathbf{I}_r)$$

Thus, $\mathbf{P} = c\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{I}_r$ holds, which proves Eqs.(10) and (11). □

Theorem 3.4 provides an efficient method to compute $(\Lambda \text{vec}(\mathbf{I}_r))$, thereby reducing redundancy significantly. Specifically, to obtain $(\Lambda \text{vec}(\mathbf{I}_r))$ using Theorem 3.4, it first takes $O(r^2 n)$ time and $O(rn)$ memory to compute $\mathbf{H} = \mathbf{V}^T\mathbf{U}\Sigma$ by Eq.(11). Due to the small size of $\mathbf{H}$ $(r \times r)$, the matrix $\mathbf{P}$ can then be iteratively computed by Eq.(11) in the low-rank r space, which requires $O(r^3)$ time and $O(r^2)$ memory in the worst case. After $\mathbf{P}$ is obtained, it takes only $O(r^2)$ time and space to compute $\Sigma\mathbf{P}\Sigma$ by Eq.(10) to get the result of $(\Lambda \text{vec}(\mathbf{I}_r))$. Thus, the total complexity of computing $(\Lambda \text{vec}(\mathbf{I}_r))$ by Theorem 3.4 is $O(r^2 n)$ time and $O(rn)$ memory $(r \ll n)$. In contrast, the traditional method to compute $(\Lambda \text{vec}(\mathbf{I}_r))$ demands $O(r^6 + r^4 n^2)$ time and $O(r^2 n^2)$ memory to prepare all entries of Λ via Eq.(6b) and Theorem 3.1. Intuitively, Theorem 3.4 does not require the entire Λ to be prepared and instead processes $(\Lambda \text{vec}(\mathbf{I}_r))$ as a whole via Eqs.(10) and (11), thus eliminating a large amount of redundancy.

4) Avoiding Tensor Product $(\mathbf{U} \otimes \mathbf{U})$ in Eq.(8). We next focus on optimising $(\mathbf{U} \otimes \mathbf{U})$ in Eq.(8) to further accelerate $\mathbf{S}$ computation since $(\mathbf{U} \otimes \mathbf{U})$, if carried out directly, consumes $O(r^2 n^2)$ time and memory, being the most expensive part that inhibits the scalability of CoSimRank on large graphs. Fortunately, by integrating Theorems 3.2–3.4, we observe that the tensor product $(\mathbf{U} \otimes \mathbf{U})$ in Eq.(8) can be effectively avoided by applying the property $(\mathbf{U} \otimes \mathbf{U})\text{vec}(\Sigma\mathbf{P}\Sigma) = \text{vec}(\mathbf{U}(\Sigma\mathbf{P}\Sigma)\mathbf{U}^T)$, thereby enabling a further significant reduction in $\mathbf{S}$ computation. To be more specific, we have the following theorem:

THEOREM 3.5. *The multi-source CoSimRank similarity $[\mathbf{S}]_{*,Q}$ over a set of queries Q in Eq.(8) can be computed efficiently as*

$$[\mathbf{S}]_{*,Q} = [\mathbf{I}_n]_{*,Q} + c\mathbf{Z}[\mathbf{U}]_{Q,*}^T \tag{12}$$

where $\mathbf{Z} := \mathbf{U}(\Sigma\mathbf{P}\Sigma)$ *satisfies Eq.*(11).

PROOF. Plugging Eq.(10) into Eq.(8) produces

$$\text{vec}(\mathbf{S}) = \text{vec}(\mathbf{I}_n) + c(\mathbf{U} \otimes \mathbf{U})\text{vec}(\Sigma\mathbf{P}\Sigma) = \text{vec}(\mathbf{I}_n) + \text{vec}(c\mathbf{U}(\Sigma\mathbf{P}\Sigma)\mathbf{U}^T)$$

Taking off $\text{vec}(*)$ operator on both sides yields $\mathbf{S} = \mathbf{I}_n + c\mathbf{U}(\Sigma\mathbf{P}\Sigma)\mathbf{U}^T$. Then, applying Theorem 3.4 to the term $(\Sigma\mathbf{P}\Sigma)$ and right-multiplying both sides by $[\mathbf{I}_n]_{*,Q}$ result in Eq.(12). □

Theorem 3.5 substantially accelerates multi-source CoSimRank search while avoiding the cost-inhibitive tensor product $(\mathbf{U} \otimes \mathbf{U})$. Using Eq.(12), it only takes $O(nr^2 + nr|Q|)$ time $(r \ll n)$ and $O(nr)$ memory to assess $[\mathbf{S}]_{*,Q}$, consisting of (i) $O(nr^2)$ time and $O(rn)$ memory to obtain $\mathbf{Z}$ and (ii) $O(nr|Q|)$ time and $O(nr)$ memory to compute $\mathbf{Z}[\mathbf{U}]_{Q,*}^T$. As an extreme case when $Q = \mathcal{V}$ (*resp.* $Q = \{q\}$), Eq.(8) reduces to all-pairs (*resp.* single-source) search. In contrast, the cost of Eq.(8), even when applied to the multi-source scenario to compute $[\mathbf{S}]_{*,Q}$, is still dominated by the $O(n^2 r^2)$ time and memory of the tensor product $(\mathbf{U} \otimes \mathbf{U})$, rather than the size of queries $|Q|$.

3.3 Putting Them All Together

Combining Theorems 3.1–3.5, we next provide a complete algorithm, namely CSR+, for efficient multi-source CoSimRank computation that scales well on billion-edge graphs, as shown in Algorithm 1.

Algorithm. CSR+ takes as inputs a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a set of queries Q $(\subseteq \mathcal{V})$, and a target low rank r $(\ll n)$, and outputs multi-source CoSimRank scores, $[\mathbf{S}]_{*,Q}$, between each node in $\mathcal{G}$ and any query in Q. CSR+ works as follows: First, the column-normalised adjacency matrix $\mathbf{Q}$ $(n \times n)$ is decomposed into $\mathbf{U}\Sigma\mathbf{V}^T$ using the low rank-r SVD (lines 1–2). Then, using $\mathbf{H}_0 = \mathbf{V}^T\mathbf{U}\Sigma$, a low-dimensional subspace $(r \times r)$ is built (line 3). In the subspace, the matrix $\mathbf{P}$ is determined iteratively using a repeated squaring approach (line 5). There is a loop (lines 4–5) that iterates until $\mathbf{P}_k$ converges to $\mathbf{P}$. To guarantee desired accuracy ϵ, this loop terminates when the number of iterations, k, reaches $\max\{0, \lceil \log_2 \log_c \epsilon \rceil + 1\}$ using the repeated squaring approach of our prior work [12]. At this point, $\|\mathbf{P}_k - \mathbf{P}\|_{\max} < \epsilon$. Using $\mathbf{P}$, $\mathbf{Z}$ is computed only once and memoised for later use (line 6). During the query stage, the similarity $[\mathbf{S}]_{*,Q}$ is computed from $\mathbf{Z}$ and $\mathbf{U}$ on an as-needed basis (line 7).

Example 3.6. Recall the graph $\mathcal{G}$ in Figure 1(a). Given a query set $Q = \{b, d\}$, low rank $r = 3$, and damping factor $c = 0.6$, CSR+ computes the CoSimRank similarities $[\mathbf{S}]_{*,Q}$ as follows:

First, the column normalised adjacency matrix $\mathbf{Q}$ is decomposed into $\mathbf{U}\Sigma\mathbf{V}^T$ via low rank-3 SVD (lines 1–2) as follows:

$$\overset{\mathbf{Q}}{\begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/2 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1/3 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/2 & 0 \end{bmatrix}} = \overset{\mathbf{U}}{\begin{bmatrix} 0.58 & 0 & 0 \\ 0 & 0.53 & -0.47 \\ 0.58 & 0 & 0 \\ 0 & 0.53 & -0.47 \\ 0 & 0.66 & 0.75 \\ 0.58 & 0 & 0 \end{bmatrix}} \overset{\Sigma}{\begin{bmatrix} 1.73 & 0 & 0 \\ 0 & 0.87 & 0 \\ 0 & 0 & 0.54 \end{bmatrix}} \overset{\mathbf{V}^T}{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0.40 & 0 & 0.58 & 0 & 0.40 & 0.58 \\ -0.58 & 0 & 0.40 & 0 & -0.58 & 0.40 \end{bmatrix}}$$

Next, from $\mathbf{U}, \Sigma, \mathbf{V}$, we obtain $\mathbf{H}_0$ (line 3) and iteratively compute $\mathbf{P}$ via repeated squaring (line 5) in the small 3×3 subspace:

$$\mathbf{H}_0 = \begin{bmatrix} 0 & 0.46 & -0.25 \\ 1.57 & 0.23 & 0.16 \\ 0.22 & -0.34 & -0.23 \end{bmatrix} \text{ and } \mathbf{P}_0 = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \xrightarrow[\text{line 5}]{\text{via}} \mathbf{P} = \begin{bmatrix} 1.49 & 0.44 & -0.24 \\ 0.44 & 3.44 & 0.04 \\ -0.24 & 0.04 & 1.30 \end{bmatrix}$$

Finally, $\mathbf{Z}$ is computed from $\mathbf{P}$ (line 6) and the multi-source CoSimRank $[\mathbf{S}]_{*,Q}$ is obtained from $\mathbf{Z}$ (line 7):

$$\overset{\mathbf{Z} = \mathbf{U}(\Sigma\mathbf{P}\Sigma)}{\begin{bmatrix} 2.57 & 0.46 & 2.57 & 0.46 & 0.27 & 2.57 \\ 0.38 & 1.38 & 0.38 & 1.38 & 1.76 & 0.38 \\ -0.13 & -0.17 & -0.13 & -0.17 & 0.30 & -0.13 \end{bmatrix}^T} \xrightarrow[\text{line 7}]{\text{via}} \overset{[\mathbf{S}]_{*,Q} = [\mathbf{I}_n]_{*,Q} + c\mathbf{Z}[\mathbf{U}]_{Q,*}^T}{\begin{bmatrix} 0.16 & 1.49 & 0.16 & 0.49 & 0.48 & 0.16 \\ 0.16 & 0.49 & 0.16 & 1.49 & 0.48 & 0.16 \end{bmatrix}^T}$$ □

Correctness. The correctness of CSR+ can be readily verified by Theorems 3.1–3.5. It can be shown that $\mathbf{S}$ returned by CSR+ is the solution to Eq.(1), yielding the same result as Li *et al.*'s method [4].

Complexity. Regarding total cost, we have the following theorem:

THEOREM 3.7. *The total computational cost of CSR+ is bounded by $O(r(m + n(r + |Q|))$ time and $O(rn)$ memory $(r \ll n)$.*

PROOF. The time and space complexities of each line of CSR+ (in Algorithm 1) are analysed in the following table:

Line #	Time	Memory	Line #	Time	Memory
1	$O(m)$	$O(m)$	4-5	$O(r^3)$	$O(r^2)$
2	$O(mr + r^3)$	$O(nr)$	6	$O(r^2 + nr^2)$	$O(nr)$
3	$O(nr^2 + nr)$	$O(nr)$	7	$O(nr\|Q\|)$	$O(nr)$

```
Algorithm 1: CSR+: Multi-Source CoSimRank ($\mathcal{G}, \mathcal{Q}, c, r, \epsilon$)
Input  : $\mathcal{G}$: a graph, $\mathcal{Q}$: a set of queries, $c$: damping factor,
         $r$: target low rank, $\epsilon$: desired accuracy (e.g. $\epsilon = 10^{-5}$)
Output : $[\mathbf{S}]_{*,\mathcal{Q}}$: multi-source CoSimRank similarity
                  I. Precomputation
1 Initialise $\mathbf{Q}$ as the column normalised adjacency matrix of $\mathcal{G}$
2 Decompose $\mathbf{Q} := \mathbf{U\Sigma V}^T$ using a low rank-$r$ SVD
3 Initialise $\mathbf{H}_0 := \mathbf{V}^T\mathbf{U\Sigma}$, $\mathbf{P}_0 := \mathbf{I}_r$, and $k := 0$
4 while $k \le \max\{0, [\log_2 \log_c \epsilon] + 1\}$ do
5     $\mathbf{P}_{k+1} := \mathbf{P}_k + c^{2^k}\mathbf{H}_k\mathbf{P}_k(\mathbf{H}_k)^T,\ \mathbf{H}_{k+1} := (\mathbf{H}_k)^2,\ k := k+1$
6 Compute the matrix $\mathbf{Z} := \mathbf{U}(\mathbf{\Sigma P}_k\mathbf{\Sigma})$
                  II. Online Multi-Source Query
7 return $[\mathbf{S}]_{*,\mathcal{Q}} := [\mathbf{I}_n]_{*,\mathcal{Q}} + c\mathbf{Z}[\mathbf{U}]^T_{\mathcal{Q},*}$
```

Since $r \ll n$, the total cost of CSR+ is dominated by $O(m + mr + 2r^3 + 2nr^2 + 3nr + rn|\mathcal{Q}|)$ time and $O(m + nr + r^2)$ space, which is bounded by $O(r(m + n(r + |\mathcal{Q}|))$ time and $O(rn)$ space. □

4 EXPERIMENTAL EVALUATION

4.1 Experimental Settings

Datasets. We use real-life datasets publicly available on SNAP [3]:

Data	$m = \|\mathcal{E}\|$	$n = \|\mathcal{V}\|$	m/n	Description
FB	88,234	4,039	21.9	Social friendship from ego-Facebook
P2P	54,705	22,687	2.4	Gnutella peer-to-peer network
YT	5,975,248	1,134,890	5.3	Youtube social network communities
WT	5,021,410	2,394,385	2.1	Wikipedia talk (communication) graph
TW	1,468,365,182	41,625,230	35.3	Twitter user-follower network
WB	1,019,903,190	118,142,155	8.6	A graph obtained by a Webbase crawler

Competitors. We contrast our CSR+ with the following rivals:

- CSR-NI: a low-rank SVD based method by Li *et al.* [4].
- CSR-IT: an iterative CoSimRank method by Rothe and Schütze [6].
- CSR-RLS: an efficient single-source similarity computing method by Kusumoto *et al.* [2] applied to CoSimRank search.

Parameters. The following settings are used by default: (1) multi-source query size $|\mathcal{Q}| = 100$; (2) damping factor $c = 0.6$; (3) target rank $r = 5$; (4) for fairness of comparison, the number of iterations k for CSR-IT and CSR-RLS is made equal to low-rank r.

Implementation. Given that our key optimisation techniques are based on matrix operations, we choose MATLAB as the primary tool for implementation due to its proficiency in handling various matrix operations. Also, MATLAB offers many optimisation packages (*e.g.* sparse SVD) that are easy to implement.

Graph Storage. For graph storage, our implementation adopts the COO (Coordinate) format in MATLAB, a sparse storage method similar to an adjacency list, that allows for significant storage savings by excluding zeros of the adjacency matrix. In this format, the adjacency matrix is represented as triples $\{(x, y, 1)\}$, where x and y denote the row and column indices (*i.e.* starting and ending vertex indices) of each edge. Sorting and grouping these triples by x results in a neighbouring list $\{x : y_1, y_2, y_3, \cdots\}$ for each node. This conversion facilitates easy access of local information for any vertex x in vertex-centric algorithms, enabling message exchange with neighbouring vertices.

All experiments are run on Ubuntu 20.04 LTS with Intel® Xeon® Gold 6226R CPU @ 2.9GHz × 64 and 256GB RAM.

4.2 Experimental Results

4.2.1 Time Efficiency. Figure 2 compares the total time of CSR+ with that of its rivals (CSR-RLS, CSR-IT and CSR-NI) on real datasets. The total time of CSR+ includes preprocessing time and query time. We notice that 1) on each dataset, CSR+ is consistently 1–3 orders of magnitude faster than all the other competitors. Particularly, CSR+ is 100x faster than CSR-NI on P2P, highlighting the effectiveness of our SVD-based method that avoids expensive Kronecker products. 2) CSR-RLS is mildly slower than CSR+ on small FB and P2P, but markedly slower on medium YT and WT. This is because there are many repeated matrix product operations in CSR-RLS, and the duplicates become more noticeable when the dataset grows larger. 3) Only CSR+ survives and scales well on large TW and WB. This is because our four-stage optimisation methods for CSR+ greatly eradicate computational redundancy and employ a repeated-squaring method in the small subspace to speed up similarity computations.

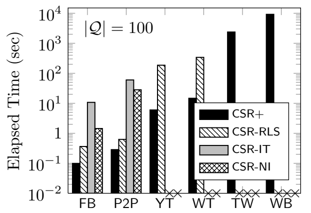


Figure 2: Total Time on Real Datasets

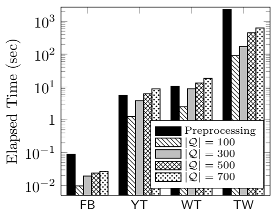
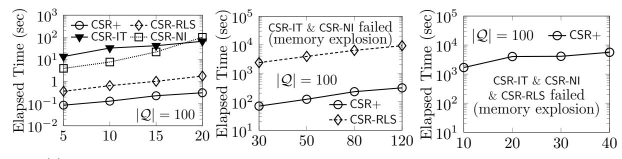
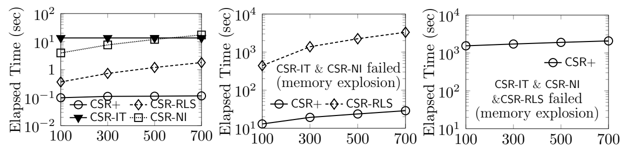


Figure 3: Preprocessing and Query Time for CSR+



(a) Vary r on FB (b) Vary r on WT (c) Vary r on TW

Figure 4: Effect of Low Rank r on CPU Time



(a) Vary $|\mathcal{Q}|$ on FB (b) Vary $|\mathcal{Q}|$ on WT (c) Vary $|\mathcal{Q}|$ on TW

Figure 5: Effect of Query Size $|\mathcal{Q}|$ on CPU Time

Figure 3 depicts the time allotted to each phase of CSR+ on real datasets when $|\mathcal{Q}|$ ranges from 100 to 700. We notice that 1) on each dataset, as $|\mathcal{Q}|$ rises, the preprocessing time of CSR+ remains unchanged (depicted by a single black bar per dataset), whereas the query time increases linearly with $|\mathcal{Q}|$. This aligns with our intuition since the offline SVD preprocessing mainly relies on graph structure, whereas the query time is proportional to $|\mathcal{Q}|$. 2) CSR+ exhibits considerably faster online query time than preprocessing time on FB and TW, due to the effective use of SVD for preprocessing. On large TW (*resp.* small FB), the query time of CSR+ is 4–25x (*resp.* 3–10x) faster than the preprocessing time. Consequently, when dealing with multiple queries on large graphs (*e.g.* TW), it is advantageous to allocate time for preprocessing to greatly reduce query time.

Figure 4 shows the impact of low rank r on time. We see that 1) as r grows, the time of CSR+, CSR-RLS and CSR-IT increases mildly, whereas the time of CSR-NI grows steeply and surpasses CSR-IT at $r = 20$. This is because CSR-NI uses costly tensor products in $O(r^4n^2)$ time, making it unscalable *w.r.t.* r and graph size. 2) On each dataset, CSR+ outperforms its rivals by 1–2 orders of magnitude as r rises. This agrees with the time complexity of CSR+ in Theorem 3.7. 3) On TW, CSR+ scales well as a result of its linear space complexity *w.r.t.* n, while other rivals wreck due to memory overload.

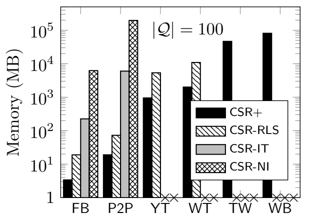


Figure 6: Total Memory on Real Datasets

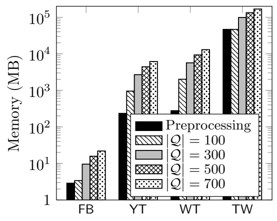
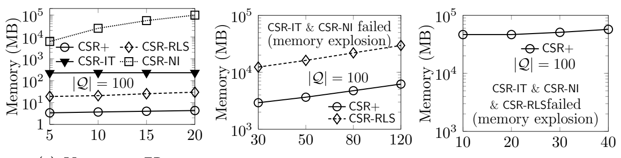


Figure 7: Preprocessing and Query Memory for CSR+



(a) Vary r on FB (b) Vary r on WT (c) Vary r on TW

Figure 8: Effect of Low Rank r on Memory

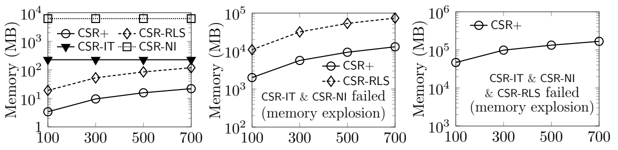
Figure 5 depicts how $|Q|$ affects time on various datasets. We discern that 1) as $|Q|$ rises, the time of CSR+ and CSR-IT is less sensitive to $|Q|$, as opposed to CSR-RLS and CSR-NI whose time grows linearly and sensitive to $|Q|$. The reason is that CSR+ has a preprocessing stage that is query-independent and dominates the total time, making the increasing time of the online query with the increase of $|Q|$ negligibly small. Since CSR-IT is an iterative algorithm to assess all node pairs, its time is orthogonal to $|Q|$. 2) When $|Q|$ is tweaked, CSR+ runs consistently 1–2 orders of magnitude faster than its rivals. On medium WT, CSR-IT and CSR-NI fail due to memory crash, while CSR+ scales well with $|Q|$, showcasing the effectiveness of our preprocessing strategies. 3) CSR-RLS is always 4–16x (*resp.* 34–115x) slower than CSR+ on FB (*resp.* WT) since CSR-RLS has many repeated computations for multiple queries. Thus, the larger $|Q|$, the higher the duplication cost of CSR-RLS, making its time more sensitive to $|Q|$.

4.2.2 *Memory Efficiency.*

Figure 6 compares the total memory usage of CSR+ with other algorithms on real datasets. We see that 1) on every dataset, the memory of CSR+ is always 1–4 orders of magnitude inferior to all of its contenders. Particularly, the memory of CSR+'s is 10,312x less than that of CSR-NI on P2P. This is because CSR+ does not memoise the results of Kronecker products but only maintains necessary low-dimensional block matrices. 2) With the increasing size of graphs, the memory of CSR+ increases linearly. This coincides with the space complexity of CSR+ as analysed in Section 3.2, implying the scalability of CSR+ on large TW and WB.

Figure 7 illustrates the memory usage for each phase of CSR+ on various datasets. We discern that 1) when the datasets become larger, the memory consumption of CSR+ increases mildly in each phase, as expected. The reason is that as the datasets expand in size, so will the decomposed matrices and resulting similarity matrix. The linearly increasing trends imply the high scalability of CSR+ on large datasets (*e.g.* TW). 2) The CSR+ memory consumed in the query phase increases linearly with the growing $|Q|$ and is always 1–46x higher than that spent during the preprocessing phase. This conforms to our intuition as the similarity matrices need to be memorised and their size rises linearly as $|Q|$ grows.

Figure 8 shows the impact of low rank r on memory for different methods. We notice that 1) CSR+ takes up 1–4 orders of magnitude less space than other rivals. This is due to its effective use of low-rank decomposition that avoids costly tensor products of CSR-NI while preventing repeated computations of CSR-RLS over multiple queries. 2) On FB, as r grows, CSR+ memory gently



(a) Vary $|Q|$ on FB (b) Vary $|Q|$ on WT (c) Vary $|Q|$ on TW

Figure 9: Effect of Query Size $|Q|$ on Memory

Data	$r = 25$	$r = 50$	$r = 100$	$r = 200$
FB	$\underline{3.3895 \times 10^{-3}}$	2.7407×10^{-3}	2.0370×10^{-3}	1.2072×10^{-3}
P2P	1.3330×10^{-4}	1.3250×10^{-4}	1.3060×10^{-4}	1.2870×10^{-4}

Table 3: Error (AvgDiff) for CSR+ and CSR-NI (underlined) on Real Datasets (FB, P2P) with $|Q| = 100$

increases, but CSR-NI memory rapidly increases since the tensor product requires huge $O(r^2 n^2)$ memory. 3) CSR-IT,CSR-NI and CSR-RLS crash on WT and TW due to memory corruption, while CSR+ survives on TW, implying the superior scalability of CSR+.

Figure 9 depicts the effect of $|Q|$ on the memory of CSR+ and its competitors on real datasets. We observe that 1) with $|Q|$ growing on each dataset, the memory of CSR+ and CSR-RLS is more sensitive to $|Q|$, whereas CSR-NI and CSR-IT stay stable when they do not fail due to memory explosion. This is because CSR+ memoises intermediate results related to the number of supplied queries on an as-needed basis. 2) The memory of CSR+ is 1–3 orders of magnitude smaller than its opponents as $|Q|$ rises. This is due to the SVD decomposition of CSR+ that achieves dimensionality reduction. 3) On TW, CSR+ scales well as $|Q|$ increases, while its rivals explode. This is because CSR+ uses the optimization technique of Theorem 3.1-3.4 and implements similarity search on low-dimensional spaces.

4.2.3 *Accuracy & Exactness.*

We next evaluate how the accuracy of CSR+ is affected by low rank r using AvgDiff, defined as $\mathrm{AvgDiff}_Q(\hat{\mathrm{S}}, \mathrm{S}) := \frac{1}{|\mathcal{V}| \times |Q|} \sum_{(i,j) \in \mathcal{V} \times Q} |[\hat{\mathrm{S}}]_{i,j} - [\mathrm{S}]_{i,j}|$, the same accuracy measure used in [4], where $\hat{\mathrm{S}}$ is the CoSimRank values of CSR+ (or CSR-NI), and S is the exact CoSimRank scores. The results on the AvgDiff of CSR+ are shown in Table 3, where an underlined number signifies the same results returned by CSR-NI as long as it survives. We notice that 1) On each dataset, when r increases, AvgDiff mildly decreases. This is comprehensible since the greater r is, the larger the subspace after low rank-r SVD decomposition, and the higher the approximation accuracy. 2) When r is reduced from 200 to 25, AvgDiff only increases by 0.218%, but the time decreases by 1–2 orders of magnitude. 3) The accuracy of CSR+ and CSR-NI is exactly the same as long as CSR-NI survives, implying that CSR+ is lossless relative to CSR-NI, which is compatible with Theorems 3.1–3.5.

5 CONCLUSIONS

This paper studies the problem of fast multi-source CoSimRank search on billion-edge graphs. We first analyse the key barriers of Li *et al.*'s method [4] and propose a four-stage optimisation scheme to eliminate costly graph tensor products and repetitive similarity calculations. On top of that, we propose a fast and scalable multi-source CoSimRank algorithm, CSR+, which reduces the time from $O(r^4 n^2)$ to $O(r(m + n(r + |Q|)))$ and memory from $O(r^2 n^2)$ to $O(rn)$, while maintaining the same accuracy as [4]. Our experiments validate that CSR+ outperforms its competitors by 1 to 4 orders of magnitude while scaling well on large graphs.

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

REFERENCES

[1] Glen Jeh and Jennifer Widom. 2002. SimRank: A measure of structural-context similarity. In *SIGKDD*. 538–543. https://doi.org/10.1145/775047.775126

[2] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *SIGMOD*. 325–336. https://doi.org/10.1145/2588555.2610526

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

[4] 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 *EDBT*. 465–476. https://doi.org/10.1145/1739041.1739098

[5] Tova Milo, Amit Somech, and Brit Youngmann. 2019. Boosting SimRank with semantics. In *EDBT*. 1–12.

[6] Sascha Rothe and Hinrich Schütze. 2014. CoSimRank: A Flexible & Efficient Graph-Theoretic Similarity Measure. In *ACL*. 1392–1402. http://aclweb.org/anthology/P/P14/P14-1131.pdf

[7] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. 2015. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences* 58, 1 (2015), 1–38.

[8] Wensi Xi, Edward A Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. 2005. Simfusion: measuring similarity using unified relationship matrix. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. 130–137.

[9] Renchi Yang. 2020. Fast Approximate CoSimRanks via Random Projections. *CoRR* abs/2010.11880 (2020). arXiv:2010.11880 https://arxiv.org/abs/2010.11880

[10] Jiale Yu, Yongliang Shen, Xinyin Ma, Chenghao Jia, Chen Chen, and Weiming Lu. 2020. SYNET: Synonym Expansion using Transitivity. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 1961–1970.

[11] Weiren Yu and Julie A McCann. 2015. Co-simmate: Quick retrieving all pairwise co-simrank scores. Association for Computational Linguistics.

[12] Weiren Yu and Julie A. McCann. 2015. Efficient Partial-Pairs SimRank Search for Large Networks. *PVLDB* 8, 5 (2015), 569–580. http://www.vldb.org/pvldb/vol8/p569-yu.pdf

[13] 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

[14] 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*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 599–608. https://doi.org/10.1145/3178876.3186126

[15] 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.

[16] Weixin Zeng, Jiuyang Tang, and Xiang Zhao. 2019. Measuring entity relatedness via entity and text joint embedding. *Neural Processing Letters* 50, 2 (2019), 1861–1875.