Noname manuscript No.
(will be inserted by the editor)

A Partition-Based Approach to Web Hyperlink Analysis

Weiren Yu · Xuemin Lin · Wenjie Zhang · Julie A. McCann

Received: date / Accepted: date

Abstract SimRank is an attractive pair-wise similarity measure based on graph structure. It iteratively follows the intuition that two nodes are assessed as similar if they are referenced by similar nodes. Real networks are often large, and links are constantly subject to minor changes. In this article, we focus on efficient dynamical computation of SimRanks on time-varying graphs. The prior approach to this problem factorizes the network via a singular value decomposition (SVD) first, and then incrementally maintains such a factorization in response to link updates at the expense of exactness. As a result, all pairs of SimRanks are updated approximately, yielding $O(r^4n^2)$ time and $O(r^2n^2)$ memory in a graph with n nodes, where r is the target rank of the low-rank SVD, but r is not negligibly small in practice.

Our solution to dynamical computation of SimRank comprises of five ingredients: (1) We first consider edge update that does not accompany new nodes insertion. We show that the SimRank update $\mathbf{\Delta S}$ in response to every link update is expressible as a rank-one Sylvester matrix equation. This provides an incremental method requiring $O(Kn^2)$ time and $O(n^2)$ memory in the worst case to update n^2 pairs of similarities for K iterations. (2) To speed up the computation further, we propose a lossless pruning strategy that captures "affected areas" of $\mathbf{\Delta S}$ to eliminate unnecessary retrieval. This reduces the time of incremental SimRank to $O(K(nd+|\mathsf{AFF}|))$, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ ($\leq n^2$) is the size of "affected areas" in $\mathbf{\Delta S}$, and in practice, $|\mathsf{AFF}| \ll n^2$. (3) We also consider edge updates that accompany node insertions, and categorize them into three cases, according to which end of the inserted edge is a new node. For each case, we devise an efficient incremental algorithm that can support new nodes insertion and accurately update affected SimRanks. (4) To achieve high memory efficiency, we formulate the SimRank changes as the sum of the outer products of two vectors, and devise a partitioning strategy that can dynamically update all pairs of SimRanks column by column in just $O(dn)$ memory. (5) We also investigate batch updates for dynamical SimRank computation, and design an efficient batch incremental method that can handle "similar sink edges" simultaneously and eliminate redundant edge updates. The experiments on various datasets demonstrate that our solution can substantially outperform the existing incremental SimRank methods, and is much faster and more memory-efficient than its competitors.

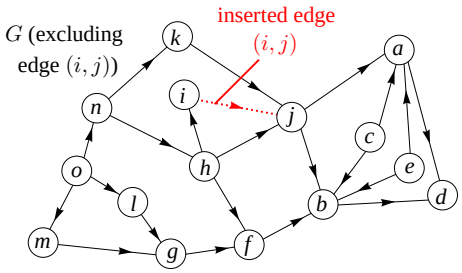
Keywords similarity search · SimRank computation · dynamical networks · optimization

1 Introduction

Recent rapid advances in web data management reveal that link analysis is becoming an important tool for similarity assessment. Due to the proliferative applications in *e.g.,* social networks, recommender systems, citation analysis, and link prediction [9], a surge of graph-based similarity measures have surfaced over the past decade. For instance, Brin and Page [2] proposed a very successful relevance measure, called Google PageRank, to rank

W. Yu · J. A. McCann
Department of Computing,
Imperial College London,
180 Queens Gate, London, UK
E-mail: weiren.yu@imperial.ac.uk

X. Lin · Wenjie Zhang
School of Computer Science and Engineering,
The University of New South Wales,
Kensington, NSW, AU
E-mail: lxue@cs.unsw.edu.au



Node-Pair	in G	in $G \cup \Delta G$	
	sim	$\mathsf{sim}_{\mathsf{true}}$	$\mathsf{sim}_{\mathsf{Li\ et\ al.}}$
(a, b)	0.075	0.062	0.073
(a, d)	0.000	0.006	0.002
$\boldsymbol{(i, f)}$	**0.246**	**0.246**	**0.246**
$\boldsymbol{(k, g)}$	**0.128**	**0.128**	**0.128**
$\boldsymbol{(k, h)}$	**0.288**	**0.288**	**0.288**
(j, f)	0.206	0.138	0.206
$\boldsymbol{(m, l)}$	**0.160**	**0.160**	**0.160**
(j, b)	0.000	0.030	0.001

Fig. 1: Incrementally update SimRanks when a new edge (i, j) (with $\{i, j\} \subseteq V$) is inserted into $G = (V, E)$

web pages. Jeh and Widom [9] devised SimRank, an appealing pair-wise similarity measure that quantifies the structural equivalence of two nodes based on link structure. Recently, Sun *et al.* [18] invented PathSim to retrieve nodes proximities in a heterogeneous graph. Among these emerging link based measures, SimRank has stood out as an attractive one in recent years, due to its simple and iterative philosophy that "two nodes are similar if they are in-linked by similar nodes", coupled with the base case that "every node is most similar to itself". This recursion not only allows SimRank to capture the global structure of a graph, but also equips SimRank with appealing mathematical insights that can inspire research in recent years. For example, Fogaras and Rácz [5] interpreted SimRank as the first meeting time of the coalescing pair-wise random walks. Li *et al.* [12] harnessed an elegant matrix equation to formulate the closed form of SimRank.

Nevertheless, the batch computation of SimRank is costly: $O(Kd'n^2)$ time for all node-pairs [20], where K is the total number of iterations, and $d' \leq d$ (d is the average in-degree of a graph). Generally, real graphs are often large, with links constantly evolving with minor changes. This is especially apparent in *e.g.,* co-citation networks, web graphs, and social networks. As a statistical example [15], there are 5%–10% links updated every week in a web graph. It is rather expensive to recompute similarities for all pairs of nodes from scratch when a graph is updated. Fortunately, we observe that when link updates are small, the affected areas for SimRank updates are often small as well. With this comes the need for incremental algorithms computing changes to SimRank in response to link updates, to discard unnecessary recomputations. Therefore, we investigate the following problem in this article.

Problem (Incremental SimRank Computation)
Given an (old) directed graph G, a small fraction of old similarities $\mathbf{S}$ in G, link changes ΔG [1] to G, and a damping factor $C \in (0, 1)$.
Retrieve the changes to the SimRank scores $\mathbf{S}$.

[1] ΔG consists of a sequence of edges to be inserted/deleted.

In comparison to the batch SimRank computation, the work on incremental SimRank updates is limited. Indeed, because of the recursive definition of SimRank, it is a big challenge to identify "affected areas" for efficiently updating SimRank in an incremental manner. To the best of our knowledge, there is only a paucity of research on incremental SimRank search. Regarding deterministic methods, Li *et al.* [12] proposed a pioneering strategy that can incrementally retrieve changes to SimRank in response to link updates. Precisely, their central idea is to factorize the backward transition matrix[2] $\mathbf{Q}$ of the original graph into $\mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ [3] via a singular value decomposition (SVD) first, and then incrementally estimate the updated matrices of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^T$ for link changes at the expense of exactness. Consequently, updating all pairs of similarities entails $O(r^4n^2)$ time and $O(r^2n^2)$ memory yet without guaranteed accuracy, where r ($\leq n$) is the target rank of the low-rank SVD approximation[4], which seems not always negligibly small in practice, as illustrated in the following example.

Example 1 Figure 1 depicts a citation graph G, a tiny fraction of DBLP, where each node is a paper, and an edge represents a reference from one paper to another. Suppose G is updated by adding an edge (i, j), denoted by ΔG (see the dash arrow). Using the damping factor $C = 0.8$ [5], we would like to compute SimRank scores in the new graph $G \cup \Delta G$.

The results are compared in the table of Figure 1, where Column '$\mathsf{sim}_{\mathsf{Li\ et\ al.}}$' denotes the approximation of SimRank scores returned by Li *et al.*'s Algorithm 3 [12], and Column '$\mathsf{sim}_{\mathsf{true}}$' denotes the "true" SimRank scores returned by a batch algorithm [6] that runs in $G \cup \Delta G$ from scratch. It can be noticed that for some node-pairs

[2] In the notation of [12], the backward transition matrix $\mathbf{Q}$ is denoted as $\tilde{\mathbf{W}}$, which is the row-normalized transpose of the adjacency matrix.

[3] Throughout this article, we use $\mathbf{X}^T$ (instead of $\tilde{\mathbf{X}}$ in [12]) to denote the transpose of matrix $\mathbf{X}$.

[4] According to [12], using our notations, $r \leq \mathrm{rank}(\mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V})$, where $\mathbf{\Delta Q}$ is the changes to $\mathbf{Q}$ for link updates.

[5] According to [9], C is empirically set around 0.6–0.8, indicating the rate of decay as similarity flows across edges.

(not highlighted in gray), the similarities obtained by Li *et al.*'s incremental method are different from the "true" SimRank scores even if the lossless SVD is used [6] during the process of updating $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$. This suggests that Li *et al.*'s incremental approach [12] is inherently *approximate*. In fact, as will be rigorously explained in Section 3, their incremental strategy would neglect some useful eigen-information whenever $\text{rank}(\mathbf{Q}) < n$.

We also notice that the target rank r for the SVD of the matrix $\mathbf{C}$ [7] is not always negligibly smaller than n. For example, in Column '$\mathsf{sim}_{\text{Li et al.}}$' of Figure 1, r is chosen to be $\text{rank}(\mathbf{C}) = 9$ to get a *lossless* SVD of $\mathbf{C}$. Although $r = 9$ is not negligibly smaller than $n = 15$, the accuracy of '$\mathsf{sim}_{\text{Li et al.}}$' is still undesirable as compared with '$\mathsf{sim}_{\text{true}}$', not to mention using $r < 9$. □

Example 1 implies that Li *et al.*'s incremental way [12] is approximate and may produce high computational cost since r is not always much smaller than n. Inspired by this, this article proposes an efficient and accurate[8] scheme for incrementally computing SimRank on link-evolving graphs. Instead of incrementally retrieving *the changes to the SVD of* $\mathbf{Q}$ for evaluating new similarities, our method can cope with the dynamic nature of a real network, by maximally reusing only a small fraction of SimRanks in an old graph and dynamically retrieving *SimRank changes* $\Delta\mathbf{S}$ *w.r.t.* link updates. Moreover, as graphs are often updated with small changes, not all pairs of similarities need recomputing. For example, in the table of Figure 1, many pairs of similarities (highlighted in gray) remain unchanged when the edge (i, j) is added. To efficiently identify "affected areas", we can express $\Delta\mathbf{S}$ as an aggregate of similarities with respect to the pairs of incoming paths, and detect the changes in these paths.

Besides, it is difficult to achieve high memory efficiency when all pairs of SimRanks are incrementally updated. This is because conventional approaches typically evaluate *all* pairs of similarities *at the same time*, and thus at least $O(n^2)$ memory is required for output. The existing work of Li *et al.* [12] entails even $O(r^2n^2)$ memory to store the intermediate results of the Kronecker product of two $n \times r$ matrices, which may become cost-inhibitive for large scale networks. Fortunately, we notice that our characterization of $\Delta\mathbf{S}$ exhibits an elegant structure, which allows all pairs of SimRanks being updated column by column. This enables a significant reduction in memory usage from quadratic to linear in the number of nodes even though all pairs of SimRanks are incrementally updated.

1.1 Main Contributions

The main contributions of this article consist of the following five ingredients:

- We first focus on unit edge update that does not accompany new nodes insertion. By characterizing the SimRank update matrix $\Delta\mathbf{S}$ *w.r.t.* every link update as a rank-one Sylvester matrix equation, we devise a fast incremental SimRank algorithm, which entails $O(Kn^2)$ time in the worst case to update n^2 pairs of similarities for K iterations.
- To speed up the computation further, we also propose an effective pruning strategy that captures "affected areas" of $\Delta\mathbf{S}$ to discard unnecessary retrieval, without loss of accuracy. This reduces the time of incremental SimRank to $O(K(nd + |\mathsf{AFF}|))$, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ ($\leq n^2$) is the size of "affected areas" in $\Delta\mathbf{S}$, and in practice, $|\mathsf{AFF}| \ll n^2$.
- We also consider edge updates that accompany new nodes insertion, and distinguish them into three categories, according to which end of the inserted edge is a new node. For each case, we devise an efficient incremental SimRank algorithm that can support new nodes insertion and accurately update affected SimRank scores.
- To achieve high memory efficiency, we next express $\Delta\mathbf{S}$ as the sum of many rank-one tensor products, and devise a novel partitioning technique that can update all pairs of SimRanks in a column-by-column style in $O(dn)$ memory, without loss of exactness.
- We also investigate the batch updates of dynamical SimRank computation. Instead of dealing with each edge update one by one, we devise an efficient algorithm that can handle a sequence of edge insertions and deletions simultaneously, by merging "similar sink edges" and minimizing unnecessary updates.
- We conduct extensive experiments on real and synthetic datasets to demonstrate that our algorithm (a) is consistently faster than the existing incremental methods from several times to over one order of magnitude; (b) is faster than its batch counterparts

[6] A *rank-α SVD* of the matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ is a factorization of the form $\mathbf{X}_\alpha = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$, where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times \alpha}$ are column-orthonormal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{\alpha \times \alpha}$ is a diagonal matrix, α is called *the target rank* of the SVD, as specified by the user. If $\alpha = \text{rank}(\mathbf{X})$, then $\mathbf{X}_\alpha = \mathbf{X}$, and we call it the *lossless SVD*. If $\alpha < \text{rank}(\mathbf{X})$, then $\|\mathbf{X} - \mathbf{X}_\alpha\|_2$ gives the least square estimate error, and we call it the *low-rank SVD*.

[7] As defined in [12], r is the target rank for the SVD of the auxiliary matrix $\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T \cdot \Delta\mathbf{Q} \cdot \mathbf{V}$, where $\Delta\mathbf{Q}$ is the changes to $\mathbf{Q}$ for link updates.

[8] Herein, the "accurate" algorithm means that its iterative result will converge to the exact SimRank solution when the number of iterations increases.

especially when link updates are small; (c) entails linear memory and scales well on large graphs even if all pairs of SimRanks are computed incrementally; (d) for batch updates, runs even faster than the repeated running of unit update algorithms.

This article is a substantial extension of our previous work [21]. We have made the following major updates: (1) In Section 6, we study three types of edge updates that accompany new nodes insertion. This solidly extends [21] and Li *et al.* 's incremental method [12] whose edge updates disallow node changes. (2) In Section 7, we propose a novel partitioning strategy that can significantly reduce the memory from $O(n^2)$ [21] to $O(dn)$ space for incrementally updating all pairs of SimRanks, with no compromise in running time and accuracy. (3) In Section 8, we also investigate batch updates for dynamic SimRank computation, and devise an efficient algorithm that can handle "similar sink edges" simultaneously and discard unnecessary unit updates further. (4) In Section 9, we also conduct additional experiments on real and synthetic datasets to verify the high memory efficiency and fast computational time of our extension methods. (5) In Section 10, we update the related work section by incorporating some state-of-the-art SimRank research that has popped up most recently.

1.2 Organization

The remainder of this article is structured as follows: Section 2 recaps the SimRank background. Section 3 explains the limitation of Li *et al.*'s incremental way [12]. Section 4 presents our dynamical method to deal with edge update that does not accompany nodes insertion. Section 5 provides our pruning strategy to reduce the running time further. Section 6 extends our method to deal with three different types of edge updates that accompany nodes insertion. Section 7 reduces the memory space. Section 8 considers batch updates for dynamical SimRank computation. Section 9 demonstrates the experimental results. The related work is in Section 10, followed by conclusions and future work in Section 11.

2 Background of SimRank

In this section, we give a broad overview of SimRank. Intuitively, the central theme behind SimRank is that "two nodes are considered as similar if their incoming neighbors are themselves similar". Based on this idea, there have emerged two widely-used SimRank models: (1) Li *et al.*'s model (*e.g.,* [6, 8, 12, 23]) and (2) Jeh and Widom's model (*e.g.,* [4,9,10,14,20]). Throughout this article, our focus is on Li *et al.*'s SimRank model since its semantics have proved in recent work [23] more meaningful than Jeh and Widom's original model. (Please refer to Remark 1 for the detailed reasons.)

2.1 Li *et al.*'s SimRank model

Given a directed graph $G = (V, E)$ with a node set V and an edge set E, let $\mathbf{Q}$ be its backward transition matrix (that is, the transpose of the column-normalized adjacency matrix), whose entry $[\mathbf{Q}]_{i,j} = 1/\text{in-degree}(i)$ if there is an edge from j to i, and 0 otherwise. Then, Li *et al.*'s SimRank matrix, denoted by $\mathbf{S}$, is defined as

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n, \quad (1)$$

where $C \in (0, 1)$ is a damping factor, which is generally taken to be 0.6–0.8, and $\mathbf{I}_n$ is an $n \times n$ identity matrix ($n = |V|$). The notation $(\star)^T$ is the matrix transpose.

2.2 Jeh and Widom's SimRank model

Jeh and Widom's SimRank model, in matrix notations, can be formulated as

$$\mathbf{S}' = \max\{C \cdot (\mathbf{Q} \cdot \mathbf{S}' \cdot \mathbf{Q}^T), \mathbf{I}_n\}, \quad (2)$$

where $\mathbf{S}'$ is called Jeh and Widom's SimRank similarity matrix, and $\max\{\mathbf{X}, \mathbf{Y}\}$ is the matrix element-wise maximum, that is, $[\max\{\mathbf{X}, \mathbf{Y}\}]_{i,j} := \max\{[\mathbf{X}]_{i,j}, [\mathbf{Y}]_{i,j}\}$.

Remark 1 The recent work by Kusumoto *et al.* [10] has showed that $\mathbf{S}$ and $\mathbf{S}'$ do not produce the same results, implying that it is ill-advised to use these two models interchangeably. Most recently, Yu and McCann [23] have showed the subtle difference of the two SimRank models from a semantic perspective further, and justified Li *et al.*'s SimRank model to be more semantically meaningful than Jeh and Widom's model in that (a) $\mathbf{S}$ can capture more pairs of self-intersecting paths that are neglected by $\mathbf{S}'$, and (b) the diagonal entries of $\mathbf{S}$ not only can guarantee that each node is maximally similar to itself, but also distinguish the relative importance of each node, unlike $\mathbf{S}'$ whose diagonals are always 1s.

3 A Fly in the Ointment in [12]

Despite the rich semantics of Li *et al.*'s SimRank model, the existing incremental approach by Li *et al.* [12] for updating SimRank does not always obtain the correct solution $\mathbf{S}$ to Eq.(1). In this section, we rigorously explain the reason — their incremental method may lose

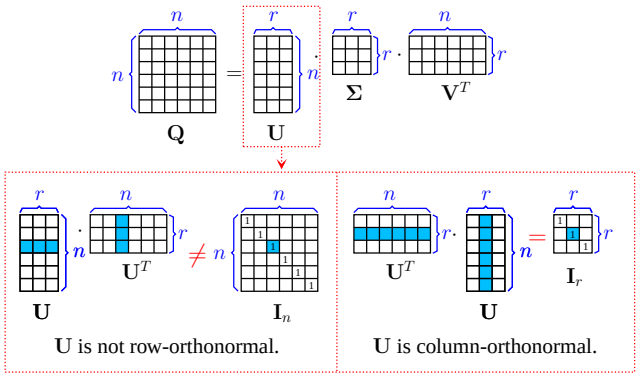


Fig. 2: $\mathbf{U}\cdot\mathbf{U}^T \neq \mathbf{I}_n$ whenever $\text{rank}(\mathbf{Q}) = r < n$

some eigen-information even if a lossless SVD is utilized for SimRank computation.

Let us first revisit the main idea of Li *et al.*'s incremental method [12]. Briefly, [12] characterizes SimRank matrix $\mathbf{S}$ in Eq.(1) in terms of three matrices $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$, where $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ are derived by the SVD of $\mathbf{Q}$, *i.e.,*

$$\mathbf{Q} = \mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T. \tag{3}$$

Then, when links are changed, [12] incrementally computes the new SimRank matrix $\tilde{\mathbf{S}}$ by updating the old matrices $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ respectively as

$$\tilde{\mathbf{U}} = \mathbf{U}\cdot\mathbf{U}_\mathbf{C}, \quad \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma}_\mathbf{C}, \quad \tilde{\mathbf{V}} = \mathbf{V}\cdot\mathbf{V}_\mathbf{C}, \tag{4}$$

[9]

where $\mathbf{U}_\mathbf{C}, \mathbf{\Sigma}_\mathbf{C}, \mathbf{V}_\mathbf{C}$ are derived from the SVD of the auxiliary matrix $\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T\cdot\mathbf{\Delta Q}\cdot\mathbf{V}$, *i.e.,*

$$\mathbf{C} = \mathbf{U}_\mathbf{C}\cdot\mathbf{\Sigma}_\mathbf{C}\cdot\mathbf{V}_\mathbf{C}{}^T, \tag{5}$$

and $\mathbf{\Delta Q}$ is the changes to $\mathbf{Q}$ in response to link updates.

However, the main problem is that the derivation of Eq.(4) rests on the assumption that

$$\mathbf{U}\cdot\mathbf{U}^T = \mathbf{V}\cdot\mathbf{V}^T = \mathbf{I}_n. \tag{6}$$

Unfortunately, Eq.(6) does *not* hold (unless $\mathbf{Q}$ is a full-rank matrix, *i.e.,* $\text{rank}(\mathbf{Q}) = n$) because in the case of $\text{rank}(\mathbf{Q}) < n$, even a "perfect" (lossless) SVD of $\mathbf{Q}$ via Eq.(3) would produce $n\times\alpha$ *rectangular* matrices $\mathbf{U}$ and $\mathbf{V}$ with $\alpha = \text{rank}(\mathbf{Q}) < n$. Thus,

$$\text{rank}(\mathbf{U}\cdot\mathbf{U}^T) = \alpha < n = \text{rank}(\mathbf{I}_n),$$

which implies that $\mathbf{U}\cdot\mathbf{U}^T \neq \mathbf{I}_n$. Similarly, $\mathbf{V}\cdot\mathbf{V}^T \neq \mathbf{I}_n$ when $\text{rank}(\mathbf{Q}) < n$. Hence, Eq.(6) is not always true, as visualized in Fig. 2.

Example 2 Consider a graph with the matrix $\mathbf{Q} = \left[\begin{smallmatrix}0 & 1\\ 0 & 0\end{smallmatrix}\right]$, and its lossless SVD:

$$\mathbf{Q} = \mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T \text{ with } \mathbf{U} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right],\ \mathbf{\Sigma} = [1],\ \mathbf{V} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right].$$

One can readily verify that

$$\mathbf{U}\cdot\mathbf{U}^T = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\cdot[1\ 0] = \left[\begin{smallmatrix}1 & 0\\ 0 & 0\end{smallmatrix}\right] \neq \left[\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right] = \mathbf{I}_n \quad (n = 2),$$

whereas

$$\mathbf{U}^T\cdot\mathbf{U} = [1\ 0]\cdot\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] = 1 = \mathbf{I}_\alpha \quad (\alpha = \text{rank}(\mathbf{Q}) = 1).$$

[10]

Thus, Eq.(6) does not hold when $\mathbf{Q}$ is not full-rank. □

To clarify why Eq.(6) gets involved in the derivation of Eq.(4), let us briefly recall from [12] the four steps of obtaining Eq.(4), and the problem lies in the last step.

STEP 1. Initially, when links are changed, the old $\mathbf{Q}$ is updated to new $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q}$. By replacing $\mathbf{Q}$ with Eq.(3), it follows that

$$\tilde{\mathbf{Q}} = \mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T + \mathbf{\Delta Q}. \tag{7}$$

STEP 2. Premultiply by $\mathbf{U}^T$ and postmultiply by $\mathbf{V}$ on both sides of Eq.(7), and then apply the property $\mathbf{U}^T\cdot\mathbf{U} = \mathbf{V}^T\cdot\mathbf{V} = \mathbf{I}_\alpha$. It follows that

$$\mathbf{U}^T\cdot\tilde{\mathbf{Q}}\cdot\mathbf{V} = \mathbf{\Sigma} + \mathbf{U}^T\cdot\mathbf{\Delta Q}\cdot\mathbf{V}. \tag{8}$$

STEP 3. Let $\mathbf{C}$ be the right-hand side of Eq.(8). Applying Eq.(5) to Eq.(8) yields

$$\mathbf{U}^T\cdot\tilde{\mathbf{Q}}\cdot\mathbf{V} = \mathbf{U}_\mathbf{C}\cdot\mathbf{\Sigma}_\mathbf{C}\cdot\mathbf{V}_\mathbf{C}{}^T. \tag{9}$$

STEP 4. Li *et al.* [12] attempted to premultiply by $\mathbf{U}$ and postmultiply by $\mathbf{V}^T$ on both sides of Eq.(9) first, and then rested on the assumption of Eq.(6) to obtain

$$\underbrace{\mathbf{U}\cdot\mathbf{U}^T}_{\overset{?}{=}\mathbf{I}_n}\cdot\tilde{\mathbf{Q}}\cdot\underbrace{\mathbf{V}\cdot\mathbf{V}^T}_{\overset{?}{=}\mathbf{I}_n} = \underbrace{(\mathbf{U}\cdot\mathbf{U}_\mathbf{C})}_{\triangleq\tilde{\mathbf{U}}}\cdot\underbrace{\mathbf{\Sigma}_\mathbf{C}}_{\triangleq\tilde{\mathbf{\Sigma}}}\cdot\underbrace{(\mathbf{V}_\mathbf{C}\cdot\mathbf{V})^T}_{\triangleq\tilde{\mathbf{V}}^T}, \tag{10}$$

which is the result of Eq.(4).

However, the problem lies in STEP 4. As mentioned before, Eq.(6) does not hold when $\text{rank}(\mathbf{Q}) < n$, which means that $\tilde{\mathbf{Q}} \neq \tilde{\mathbf{U}}\cdot\tilde{\mathbf{\Sigma}}\cdot\tilde{\mathbf{V}}^T$ in Eq.(10). Consequently, updating the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ via Eq.(4) may produce an error (up to $\|\mathbf{I}_n - \mathbf{U}\cdot\mathbf{U}^T\|_2 = 1$, which is not practically small) for incrementally "approximating" $\mathbf{S}$.

Example 3 Recall the old $\mathbf{Q}$ and its SVD in Example 2. Suppose there is a new edge insertion, associated with $\mathbf{\Delta Q} = \left[\begin{smallmatrix}0 & 0\\ 1 & 0\end{smallmatrix}\right]$. [12] first computes auxiliary matrix $\mathbf{C}$ as

$$\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T\cdot\mathbf{\Delta Q}\cdot\mathbf{V} = [1] + [1\ 0]\cdot\left[\begin{smallmatrix}0 & 0\\ 1 & 0\end{smallmatrix}\right]\cdot\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right] = [1].$$

Then, the matrix $\mathbf{C}$ is decomposed via Eq.(5) into

$$\mathbf{C} = \mathbf{U}_\mathbf{C}\cdot\mathbf{\Sigma}_\mathbf{C}\cdot\mathbf{V}_\mathbf{C}{}^T \text{ with } \mathbf{U}_\mathbf{C} = \mathbf{\Sigma}_\mathbf{C} = \mathbf{V}_\mathbf{C} = [1].$$

Finally, [12] updates the new SVD of $\tilde{\mathbf{Q}}$ via Eq.(4) as

$$\tilde{\mathbf{U}} = \mathbf{U}\cdot\mathbf{U}_\mathbf{C} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \quad \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma}_\mathbf{C} = [1], \quad \tilde{\mathbf{V}} = \mathbf{V}\cdot\mathbf{V}_\mathbf{C} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right].$$

Symbol	Description
n	the number of nodes in old graph G
d_i	in-degree of node i in old graph G
d	average in-degree of graph G
C	damping factor ($0 < C < 1$)
k	iteration number
$\mathbf{e}_i$	$n \times 1$ unit vector with a 1 in the i-th entry and 0s elsewhere
$\mathbf{Q}/\tilde{\mathbf{Q}}$	old/new (backward) transition matrix
$\mathbf{S}/\tilde{\mathbf{S}}$	old/new SimRank matrix
$\mathbf{I}_n$	$n \times n$ identity matrix
$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$
$[\mathbf{X}]_{i,\star}$	i-th row of matrix $\mathbf{X}$
$[\mathbf{X}]_{\star,j}$	j-th column of matrix $\mathbf{X}$
$[\mathbf{X}]_{i,j}$	(i,j)-th entry of matrix $\mathbf{X}$

Table 1: Symbol and Description

However, one can readily verify that

$$\tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right] \neq \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right] = \mathbf{Q} + \mathbf{\Delta Q} = \tilde{\mathbf{Q}}.$$

In comparison, a "true" SVD of $\tilde{\mathbf{Q}}$ should be

$$\tilde{\mathbf{Q}} = \hat{\mathbf{U}} \cdot \hat{\mathbf{\Sigma}} \cdot \hat{\mathbf{V}}^T \text{ with } \hat{\mathbf{U}} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right],\ \hat{\mathbf{\Sigma}} = \hat{\mathbf{V}} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right].$$

Besides, the approximation error is not small in practice

$$\|\tilde{\mathbf{Q}} - \tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T\|_2 = \|\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right] - \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]\|_2 = 1. \quad \square$$

Our analysis suggests that, only when (i) $\mathbf{Q}$ is full-rank, and (ii) the SVD of $\mathbf{Q}$ is lossless ($n = \text{rank}(\mathbf{Q}) = \alpha$), Li *et al.*'s incremental way [12] can produce the *exact* $\mathbf{S}$, but the time complexity of [12], $O(r^4n^2)$, would become $O(n^6)$, which is prohibitively expensive. In practice, as evidenced by our statistical experiments in Fig.8 on Stanford Large Network Datasets (SNAP), most real graphs are not full-rank, highlighting our need to devise an efficient method for dynamic SimRank computation.

4 Edge Update without Nodes Insertions

In this section, we consider edge update that does not accompany new nodes insertions, *i.e.,* the insertion[11] of new edge (i, j) into $G = (V, E)$ with $i \in V$ and $j \in V$. In this case, the new SimRank matrix $\tilde{\mathbf{S}}$ and the old one $\mathbf{S}$ are of the same size. As such, it makes sense to denote the SimRank change $\mathbf{\Delta S}$ as $\tilde{\mathbf{S}} - \mathbf{S}$. [12]

Table 1 lists the notations often used in this article. Below we first present the big picture of our main idea, and then get down to rigorous justifications and proofs.

[11] Due to many commonalities of "insertion" and "deletion", we will mainly focus on "insertion" here, and briefly summarize "deletion" in Subsection 4.4.

[12] As will be seen in Section 6, the inserted edge (i, j) that accompany nodes insertion cannot keep the same size of the new $\tilde{\mathbf{S}}$ and old $\mathbf{S}$. Thus, $\tilde{\mathbf{S}} - \mathbf{S}$ makes no sense in such cases.

4.1 The main idea

For each edge (i, j) insertion, we can show that $\mathbf{\Delta Q}$ is a *rank-one* matrix, *i.e.,* there exist two column vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{n\times 1}$ such that $\mathbf{\Delta Q} \in \mathbb{R}^{n\times n}$ can be decomposed into the *outer product*[13] of $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\mathbf{\Delta Q} = \mathbf{u} \cdot \mathbf{v}^T.^{14} \tag{11}$$

Based on Eq.(11), we then have an opportunity to efficiently compute $\mathbf{\Delta S}$, by characterizing it as

$$\mathbf{\Delta S} = \mathbf{M} + \mathbf{M}^T, \tag{12}$$

where the auxiliary matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ satisfies the following *rank-one* Sylvester equation:

$$\mathbf{M} = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{M} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{u} \cdot \mathbf{w}^T. \tag{13}$$

Here, $\mathbf{u}, \mathbf{w}$ are two obtainable column vectors: $\mathbf{u}$ can be derived from Eq.(11), and $\mathbf{w}$ can be described by the old $\mathbf{Q}$ and $\mathbf{S}$ (we will provide their exact expressions later after some discussions); and $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q}$.

Thus, computing $\mathbf{\Delta S}$ boils down to solving $\mathbf{M}$ in Eq.(13). The main advantage of solving $\mathbf{M}$ via Eq.(13), as compared to directly computing the new scores $\tilde{\mathbf{S}}$ via SimRank formula

$$\tilde{\mathbf{S}} = C \cdot \tilde{\mathbf{Q}} \cdot \tilde{\mathbf{S}} \cdot \tilde{\mathbf{Q}}^T + (1 - C) \cdot \mathbf{I}_n, \tag{14}$$

is the high computational efficiency. More specifically, solving $\tilde{\mathbf{S}}$ via Eq.(14) needs expensive *matrix-matrix* multiplications, whereas solving $\mathbf{M}$ via Eq.(13) involves only *matrix-vector* and *vector-vector* multiplications, which is a substantial improvement achieved by our observation that $(C \cdot \mathbf{uw}^T) \in \mathbb{R}^{n\times n}$ in Eq.(13) is a *rank-one* matrix, as opposed to the (full) *rank-n* matrix $(1-C)\cdot\mathbf{I}_n$ in Eq.(14). To further elaborate on this, we can readily convert the recursive forms of Eqs.(13) and (14), respectively, into the series forms: [15]

$$\mathbf{M} = \sum\nolimits_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{u} \cdot \mathbf{w}^T \cdot (\tilde{\mathbf{Q}}^T)^k, \tag{15}$$

$$\tilde{\mathbf{S}} = (1 - C) \cdot \sum\nolimits_{k=0}^{\infty} C^k \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{I}_n \cdot (\tilde{\mathbf{Q}}^T)^k. \tag{16}$$

To compute the sums in Eq.(15) for $\mathbf{M}$, a conventional way is to memoize $\mathbf{M}_0 \leftarrow C \cdot \mathbf{u} \cdot \mathbf{w}^T$ first (where the intermediate result $\mathbf{M}_0$ is an $n \times n$ matrix), and then iterate as

$$\mathbf{M}_{k+1} \leftarrow \mathbf{M}_0 + C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{M}_k \cdot \tilde{\mathbf{Q}}^T, \quad (k = 0, 1, 2, \cdots)$$

[13] The *outer product* of the vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{n\times 1}$ is an $n \times n$ rank-1 matrix $\mathbf{x} \cdot \mathbf{y}^T$, in contrast with the *inner product* $\mathbf{x}^T \cdot \mathbf{y}$, which is a scalar.

[15] One can readily verify that if $\mathbf{X} = \sum_{k=0}^{\infty} \mathbf{A}^k \cdot \mathbf{C} \cdot \mathbf{B}^k$ is a convergent matrix series, it is the solution of the Sylvester equation $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$.

which involves expensive *matrix-matrix* multiplications (*e.g.,* $\tilde{\mathbf{Q}} \cdot \mathbf{M}_k$). In contrast, our trick takes advantage of the *rank-one* structure of $\mathbf{u} \cdot \mathbf{w}^T$ to compute the sums in Eq.(15) for $\mathbf{M}$, by converting the conventional *matrix-matrix* multiplications $\tilde{\mathbf{Q}} \cdot (\mathbf{u}\mathbf{w}^T) \cdot \tilde{\mathbf{Q}}^T$ into only *matrix-vector* and *vector-vector* multiplications $(\tilde{\mathbf{Q}}\mathbf{u}) \cdot (\tilde{\mathbf{Q}}\mathbf{w})^T$. To be specific, we leverage two auxiliary vectors $\boldsymbol{\xi}_k, \boldsymbol{\eta}_k$, and iteratively compute Eq.(15) as follows:

$$\begin{array}{l} \text{initialize } \boldsymbol{\xi}_0 \leftarrow C \cdot \mathbf{u}, \quad \boldsymbol{\eta}_0 \leftarrow \mathbf{w}, \quad \mathbf{M}_0 \leftarrow C \cdot \mathbf{u} \cdot \mathbf{w}^T \\ \text{for } k = 0, 1, 2, \cdots \\ \quad \boldsymbol{\xi}_{k+1} \leftarrow C \cdot \tilde{\mathbf{Q}} \cdot \boldsymbol{\xi}_k, \quad \boldsymbol{\eta}_{k+1} \leftarrow \tilde{\mathbf{Q}} \cdot \boldsymbol{\eta}_k \\ \quad \mathbf{M}_{k+1} \leftarrow \boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T + \mathbf{M}_k \end{array}$$

so that *matrix-matrix* multiplications are safely avoided.

Remark 2 It is worth mentioning that our above trick is solely suitable for efficiently computing $\mathbf{M}$ in Eq.(15), but not applicable to accelerating $\tilde{\mathbf{S}}$ computation in Eq.(16). This is because $\mathbf{I}_n$ is a (full) rank-n matrix that cannot be decomposed into the outer product of two vectors. Thus, our trick is particularly tailored for improving the *incremental* computation of $\boldsymbol{\Delta}\mathbf{S}$ via Eq.(13), rather than the *batch* computation of $\tilde{\mathbf{S}}$ via Eq.(14).

4.2 Describing $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in Eqs.(11) and (13)

To compute $\boldsymbol{\Delta}\mathbf{S}$, we are going to address two problems: One is to obtain the vectors $\mathbf{u}, \mathbf{v}$ in Eq.(11) from the *rank-one* decomposition of $\boldsymbol{\Delta}\mathbf{Q}$. The other task is the description of the vector $\mathbf{w}$ in Eq.(13) in terms of the old matrices $\mathbf{Q}$ and $\mathbf{S}$, in order to guarantee that Eq.(13) is a *rank-one* Sylvester equation.

To obtain $\mathbf{u}$ and $\mathbf{v}$ in Eq.(11) with fairly cheap cost, we have the following theorem.

Theorem 1 *Given an old digraph $G = (V, E)$, if there is a new edge (i, j) with $i \in V$ and $j \in V$ to be added to G, then the change to $\mathbf{Q}$ is an $n \times n$ rank-one matrix, i.e., $\boldsymbol{\Delta}\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$, where*

$$\mathbf{u} = \begin{cases} \mathbf{e}_j & (d_j = 0) \\ \frac{1}{d_j+1}\mathbf{e}_j & (d_j > 0) \end{cases}, \quad \mathbf{v} = \begin{cases} \mathbf{e}_i & (d_j = 0) \\ \mathbf{e}_i - [\mathbf{Q}]_{j,\star}^T & (d_j > 0) \end{cases} \tag{17}$$

Proof We show this by considering the two cases below:

(i) If $d_j = 0$, then $[\mathbf{Q}]_{j,\star} = \mathbf{0}$, and the inserted edge (i, j) will update $[\mathbf{Q}]_{j,i}$ from 0 to 1, *i.e.,* $\boldsymbol{\Delta}\mathbf{Q} = \mathbf{e}_j \mathbf{e}_i^T$.

(ii) If $d_j > 0$, then all nonzeros in old $[\mathbf{Q}]_{j,\star}$ are $\frac{1}{d_j}$. The inserted edge (i, j) will update $[\mathbf{Q}]_{j,\star}$ via 2 steps: first, all nonzeros in $[\mathbf{Q}]_{j,\star}$ are changed from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$; then, the entry $[\mathbf{Q}]_{j,i}$ is changed from 0 to $\frac{1}{d_j+1}$.

$$[\tilde{\mathbf{Q}}]_{j,\star} = \tfrac{d_j}{d_j+1}[\mathbf{Q}]_{j,\star} + \tfrac{1}{d_j+1}\mathbf{e}_i^T = [\mathbf{Q}]_{j,\star} + \tfrac{1}{d_j+1}(\mathbf{e}_i^T - [\mathbf{Q}]_{j,\star})$$

Since only the j-th row of $\mathbf{Q}$ is affected, it follows that

$$\tilde{\mathbf{Q}} - \mathbf{Q} = \underbrace{\tfrac{1}{d_j+1}\mathbf{e}_j}_{:=\mathbf{u}} \underbrace{(\mathbf{e}_i^T - [\mathbf{Q}]_{j,\star})}_{:=\mathbf{v}^T} = \mathbf{u} \cdot \mathbf{v}^T$$

Finally, combining (i) and (ii), Eq.(17) holds. □

Example 4 Recall the digraph G in Fig. 1, and the edge (i, j) to be inserted into G. Notice that, in the old G, $d_j = 2 > 0$ and

$$[\mathbf{Q}]_{j,\star} = \big[\, 0 \ \cdots \ 0 \ \overset{(h)}{\tfrac{1}{2}} \ 0 \ 0 \ \overset{(k)}{\tfrac{1}{2}} \ 0 \ \cdots \ 0 \,\big] \in \mathbb{R}^{1\times 15}.$$

According to Theorem 1, the change $\boldsymbol{\Delta}\mathbf{Q}$ is a 15×15 rank-one matrix, and can be decomposed as $\mathbf{u} \cdot \mathbf{v}^T$ with

$$\mathbf{u} = \tfrac{1}{d_j+1}\mathbf{e}_j = \tfrac{1}{3}\mathbf{e}_j = \big[\, 0 \ \cdots \ 0 \ \overset{(j)}{\tfrac{1}{3}} \ 0 \ \cdots \ 0 \,\big]^T \in \mathbb{R}^{15\times 1},$$

$$\mathbf{v} = \mathbf{e}_i - [\mathbf{Q}]_{j,\star}^T = \big[\, 0 \ \cdots \ 0 \ \overset{(h)}{-\tfrac{1}{2}} \ \overset{(i)}{1} \ \overset{(j)}{0} \ \overset{(k)}{-\tfrac{1}{2}} \ 0 \ \cdots \ 0 \,\big]^T \in \mathbb{R}^{15\times 1}. \quad \square$$

Theorem 1 suggests that the change $\boldsymbol{\Delta}\mathbf{Q}$ is an $n \times n$ *rank-one* matrix, which can be obtain in only constant time from d_j and $[\mathbf{Q}]_{j,\star}^T$. In light of this, we next describe $\mathbf{w}$ in Eq.(13) in terms of the old $\mathbf{Q}$ and $\mathbf{S}$ such that Eq.(13) is a *rank-one* Sylvester equation.

Theorem 2 *Suppose there is a new edge (i, j) with $i \in V$ and $j \in V$ to be inserted to G. Let $\mathbf{u}$ and $\mathbf{v}$ be the rank-one decomposition of $\boldsymbol{\Delta}\mathbf{Q}$ via Theorem 1. Then,*

(i) there exists a vector $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ with

$$\mathbf{y} = \mathbf{Q} \cdot \mathbf{z}, \quad \lambda = \mathbf{v}^T \cdot \mathbf{z}, \quad \mathbf{z} = \mathbf{S} \cdot \mathbf{v} \tag{18}$$

such that Eq.(13) *is the* rank-one *Sylvester equation.*

(ii) Utilizing the solution $\mathbf{M}$ to Eq.(13)*, the SimRank update matrix $\boldsymbol{\Delta}\mathbf{S}$ can be represented by Eq.*(12)*.*

Proof We show this by following the two steps:

(a) We first formulate $\boldsymbol{\Delta}\mathbf{S}$ in a recursive style. To describe $\boldsymbol{\Delta}\mathbf{S}$ in terms of the old $\mathbf{Q}$ and $\mathbf{S}$, we subtract Eq.(1) from Eq.(14), and apply $\boldsymbol{\Delta}\mathbf{S} = \tilde{\mathbf{S}} - \mathbf{S}$, yielding

$$\boldsymbol{\Delta}\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \tilde{\mathbf{Q}} \cdot \boldsymbol{\Delta}\mathbf{S} \cdot \tilde{\mathbf{Q}}^T - C \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T. \tag{19}$$

By Theorem 1, there are two vectors $\mathbf{u}$ and $\mathbf{v}$ such that

$$\tilde{\mathbf{Q}} = \mathbf{Q} + \boldsymbol{\Delta}\mathbf{Q} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T. \tag{20}$$

Then, we plug Eq.(20) into the term $C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T$ of Eq.(19), and simplify the result into

$$\boldsymbol{\Delta}\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \boldsymbol{\Delta}\mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{T} \tag{21}$$

$$\text{with } \mathbf{T} = \mathbf{u}(\mathbf{Q}\mathbf{S}\mathbf{v})^T + (\mathbf{Q}\mathbf{S}\mathbf{v})\mathbf{u}^T + (\mathbf{v}^T\mathbf{S}\mathbf{v})\mathbf{u}\mathbf{u}^T. \tag{22}$$

We can verify that $\mathbf{T}$ is a symmetric matrix ($\mathbf{T} = \mathbf{T}^T$). Moreover, we note that $\mathbf{T}$ is the sum of two rank-one matrices. This can be verified by letting

$$\mathbf{z} \triangleq \mathbf{S} \cdot \mathbf{v}, \ \mathbf{y} \triangleq \mathbf{Q} \cdot \mathbf{z}, \ \lambda \triangleq \mathbf{v}^T \cdot \mathbf{z}.$$

Then, using the auxiliary vectors $\mathbf{z}, \mathbf{y}$ and the scalar λ, we can simplify Eq.(22) into

$$\mathbf{T} = \mathbf{u} \cdot \mathbf{w}^T + \mathbf{w} \cdot \mathbf{u}^T, \quad \text{with } \mathbf{w} = \mathbf{y} + \tfrac{\lambda}{2}\mathbf{u}. \tag{23}$$

(b) We next convert the recursive form of $\Delta\mathbf{S}$ into the series form. One can readily verify that

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C} \quad \Leftrightarrow \quad \mathbf{X} = \sum\nolimits_{k=0}^{\infty} \mathbf{A}^k \cdot \mathbf{C} \cdot \mathbf{B}^k \tag{24}$$

Thus, based on Eq.(24), the recursive definition of $\Delta\mathbf{S}$ in Eq.(21) naturally leads itself to the series form:

$$\Delta\mathbf{S} = \sum\nolimits_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{T} \cdot (\tilde{\mathbf{Q}}^T)^k.$$

Combining this with Eq.(23) yields

$$\begin{aligned} \Delta\mathbf{S} &= \sum\nolimits_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \left(\mathbf{u} \cdot \mathbf{w}^T + \mathbf{w} \cdot \mathbf{u}^T\right) \cdot (\tilde{\mathbf{Q}}^T)^k \\ &= \mathbf{M} + \mathbf{M}^T \quad \text{with } \mathbf{M} \text{ being defined in Eq.(15).} \end{aligned}$$

By Eq.(24), the series form of $\mathbf{M}$ in Eq.(15) satisfies the rank-one Sylvester recursive form of Eq.(13). □

Theorem 2 provides an elegant expression of $\mathbf{w}$ in Eq.(13). To be precise, given $\mathbf{Q}$ and $\mathbf{S}$ in the old graph G, and an edge (i, j) inserted to G, one can find $\mathbf{u}$ and $\mathbf{v}$ via Theorem 1 first, and then resort to Theorem 2 to compute $\mathbf{w}$ from $\mathbf{u}, \mathbf{v}, \mathbf{Q}, \mathbf{S}$. Due to the existence of the vector $\mathbf{w}$, it can be guaranteed that the Sylvester equation (13) is *rank-one*. Henceforth, our aforementioned trick can be employed to iteratively compute $\mathbf{M}$ in Eq.(15), needing no *matrix-matrix* multiplications.

4.3 Characterizing $\Delta\mathbf{S}$

Obtaining $\mathbf{w}$ from Theorem 2 is intended to speed up the computation of $\Delta\mathbf{S}$. Indeed, when edge $(i, j)_{i \in V, j \in V}$ is added, the *whole* process of computing $\Delta\mathbf{S}$ in Eq.(12), given $\mathbf{Q}$ and $\mathbf{S}$, needs no *matrix-matrix* multiplications. Precisely, the computation of $\Delta\mathbf{S}$ consists of two phases: (i) Given $\mathbf{Q}$ and $\mathbf{S}$, we compute $\mathbf{w}$ from Theorems 1 and 2. This phase includes only the matrix-vector multiplications (*e.g.,* $\mathbf{Qz}, \mathbf{Sv}$), the inner product of vectors (*e.g.,* $\mathbf{v}^T\mathbf{z}$), and the vector scaling and additions, *i.e.,* SAXPY (*e.g.,* $\mathbf{y} + \frac{\lambda}{2}\mathbf{u}$). (ii) Given $\mathbf{w}$, we compute $\mathbf{M}$ via Eq.(15). In this phase, our novel iterative model for Eq.(15) can circumvent the *matrix-matrix* multiplications. Thus, taking (i) and (ii) together, it suffices to use only the *matrix-vector* and *vector-vector* operations in the whole process of $\Delta\mathbf{S}$ computation.

Leveraging Theorems 1 and 2, we next characterize the SimRank change $\Delta\mathbf{S}$.

Theorem 3 *If there is a new edge (i, j) with $i \in V$ and $j \in V$ to be inserted to G, then the SimRank change $\Delta\mathbf{S}$ can be characterized as*

$$\Delta\mathbf{S} = \mathbf{M} + \mathbf{M}^T \quad \textit{with}$$

$$\mathbf{M} = \sum\nolimits_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{e}_j \cdot \boldsymbol{\gamma}^T \cdot (\tilde{\mathbf{Q}}^T)^k, \tag{25}$$

where the auxiliary vector $\boldsymbol{\gamma}$ is obtained as follows:

(i) when $d_j = 0$,

$$\boldsymbol{\gamma} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \tfrac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j \tag{26}$$

(ii) when $d_j > 0$,

$$\boldsymbol{\gamma} = \tfrac{1}{(d_j+1)} \left(\mathbf{Q}[\mathbf{S}]_{\star,i} - \tfrac{1}{C}[\mathbf{S}]_{\star,j} + (\tfrac{\lambda}{2(d_j+1)} + \tfrac{1}{C} - 1)\mathbf{e}_j\right) \tag{27}$$

and the scalar λ can be derived from

$$\lambda = [\mathbf{S}]_{i,i} + \tfrac{1}{C} \cdot [\mathbf{S}]_{j,j} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} - \tfrac{1}{C} + 1. \tag{28}$$

Proof We divide the proof into the following two cases:

(i) When $d_j = 0$, according to Eq.(17) in Theorem 1, $\mathbf{v} = \mathbf{e}_i$, $\mathbf{u} = \mathbf{e}_j$. Plugging them into Eq.(18) gets

$$\mathbf{z} = [\mathbf{S}]_{\star,i}, \quad \mathbf{y} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i}, \quad \lambda = [\mathbf{S}]_{i,i}.$$

Thus, applying $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ in Theorem 2, we have

$$\mathbf{w} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \tfrac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j.$$

Coupling this with Eq.(15), $\mathbf{u} = \mathbf{e}_j$, and Theorem 2 completes the proof of the case $d_j = 0$ for Eq.(26).

(ii) When $d_j > 0$, Eq.(17) in Theorem 1 implies that

$$\mathbf{v} = \mathbf{e}_i - [\mathbf{Q}]_{j,\star}^T, \quad \mathbf{u} = \tfrac{1}{d_j+1} \cdot \mathbf{e}_j. \tag{29}$$

Substituting these back into Eq.(18) yields

$$\begin{aligned} \mathbf{z} &= [\mathbf{S}]_{\star,i} - \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T, \quad \mathbf{y} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T, \\ \lambda &= [\mathbf{S}]_{i,i} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} + [\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T. \end{aligned}$$

To simplify $\mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T$ in $\mathbf{y}$, and $[\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T$ in λ, we postmultiply both sides of Eq.(1) by $\mathbf{e}_j$ to obtain

$$\mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T = \tfrac{1}{C} \cdot ([\mathbf{S}]_{\star,j} - (1 - C) \cdot \mathbf{e}_j). \tag{30}$$

We also premultiply both sides of Eq.(30) by $\mathbf{e}_j^T$ to get

$$[\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T = \tfrac{1}{C} \cdot ([\mathbf{S}]_{j,j} - 1) + 1. \tag{31}$$

Plugging Eqs.(30) and (31) into $\mathbf{y}$ and λ, respectively, and then putting $\mathbf{y}$ and λ into $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ produce

$$\mathbf{w} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \tfrac{1}{C} \cdot [\mathbf{S}]_{\star,j} + (\tfrac{1}{C} + \tfrac{\lambda}{2(d_j+1)} - 1) \cdot \mathbf{e}_j,$$

where $\lambda = [\mathbf{S}]_{i,i} + \frac{1}{C} \cdot [\mathbf{S}]_{j,j} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} - \frac{1}{C} + 1$.

Combining this with Eqs.(15) and (29) shows the case $d_j > 0$ for Eq.(27).

Finally, taking (i) and (ii) together with Theorem 2 completes the entire proof. □

Theorem 3 provides an efficient method to compute the incremental SimRank matrix $\mathbf{\Delta S}$, by utilizing the previous information of $\mathbf{Q}$ and $\mathbf{S}$, as opposed to [12] that needs to maintain the incremental SVD.

To achieve even higher efficiency for computing $\mathbf{\Delta S}$ by Theorem 3, two extra tricks are worth mentioning: (i) Note that, by viewing the matrix $\mathbf{Q}$ as a stack of row vectors, the j-th row of the term $(\mathbf{Q}\cdot[\mathbf{S}]_{\star,i})$ in Eqs.(26) and (27) is the inner product $[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}$, which is the term in Eq.(28). Thus, the resulting $[\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}]_{j,\star}$, once computed, can be reused to compute $[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}$ in λ. (ii) As suggested earlier, computing the matrix series for $\mathbf{M}$ needs no matrix-matrix multiplications, but involves the matrix-vector multiplications (*e.g.,* $\boldsymbol{\eta}_{k+1}\leftarrow\tilde{\mathbf{Q}}\cdot\boldsymbol{\eta}_k$). Since $\tilde{\mathbf{Q}}=\mathbf{Q}+\mathbf{u}\cdot\mathbf{v}^T$ via Theorem 1, we notice that $\tilde{\mathbf{Q}}\cdot\boldsymbol{\eta}_k$ can be computed more efficiently, with no need to memoize $\tilde{\mathbf{Q}}$ in extra memory space, as follows:

$$\tilde{\mathbf{Q}}\cdot\boldsymbol{\eta}_k=\mathbf{Q}\cdot\boldsymbol{\eta}_k+(\mathbf{v}^T\cdot\boldsymbol{\eta}_k)\cdot\mathbf{u}.$$

4.4 Deleting an edge $(i,j)_{i\in V,\ j\in V}$ from $G=(V,E)$

For an edge deletion, we next propose a Theorem 3-like technique that can efficiently update SimRanks.

Theorem 4 *When an edge $(i,j)_{i\in V,\ j\in V}$ is deleted from $G=(V,E)$, the changes to $\mathbf{Q}$ is a rank-one matrix, which can be described as $\mathbf{\Delta Q}=\mathbf{u}\cdot\mathbf{v}^T$, where*

$$\mathbf{u}=\begin{cases}\mathbf{e}_j & (d_j=1)\\ \frac{1}{d_j-1}\mathbf{e}_j & (d_j>1)\end{cases},\quad \mathbf{v}=\begin{cases}-\mathbf{e}_i & (d_j=1)\\ [\mathbf{Q}]^T_{j,\star}-\mathbf{e}_i & (d_j>1)\end{cases}$$

The changes $\mathbf{\Delta S}$ to SimRank can be characterized as

$$\mathbf{\Delta S}=\mathbf{M}+\mathbf{M}^T\ \ \textit{with}\ \mathbf{M}=\sum\nolimits_{k=0}^{\infty}C^{k+1}\tilde{\mathbf{Q}}^k\mathbf{e}_j\boldsymbol{\gamma}^T(\tilde{\mathbf{Q}}^T)^k,$$

where the auxiliary vector $\boldsymbol{\gamma}:=$

$$\begin{cases}-\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}+\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j & (d_j=1)\\ \frac{1}{(d_j-1)}\left(\frac{1}{C}\cdot[\mathbf{S}]_{\star,j}-\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}+(\frac{\lambda}{2(d_j-1)}-\frac{1}{C}+1)\cdot\mathbf{e}_j\right) & (d_j>1)\end{cases}$$

and $\lambda:=[\mathbf{S}]_{i,i}+\frac{1}{C}\cdot[\mathbf{S}]_{j,j}-2\cdot[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}-\frac{1}{C}+1.$ □

(The proof is similar to those of Theorems 1–3, and is omitted due to space limitations.)

4.5 Algorithm

We next present an efficient incremental SimRank algorithm, denoted as Inc-uSR, that supports the edge insertion without accompanying new nodes insertion.

Given an old graph $G=(V,E)$, a new edge (i,j) with $i\in V$ and $j\in V$ to be inserted to G, the old similarities $\mathbf{S}$ in G, and the damping factor C, Inc-uSR incrementally computes $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$ as follows:

Algorithm 1: Inc-uSR $(G,(i,j),\mathbf{S},K,C)$

Input : a directed graph $G=(V,E)$,
a new edge $(i,j)_{i\in V,\ j\in V}$ inserted to G,
the old similarities $\mathbf{S}$ in G,
the number of iterations K,
the damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$.

1 initialize the transition matrix $\mathbf{Q}$ in G ;
2 $d_j:=$ in-degree of node j in G ;
3 memoize $\mathbf{w}:=\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}$;
4 compute $\lambda:=[\mathbf{S}]_{i,i}+\frac{1}{C}\cdot[\mathbf{S}]_{j,j}-2\cdot[\mathbf{w}]_j-\frac{1}{C}+1$;
5 **if** $d_j=0$ **then**
6 $\quad\mathbf{u}:=\mathbf{e}_j,\ \mathbf{v}:=\mathbf{e}_i,\ \boldsymbol{\gamma}:=\mathbf{w}+\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j$;
7 **else**
8 $\quad\mathbf{u}:=\frac{1}{d_j+1}\mathbf{e}_j,\quad\mathbf{v}:=\mathbf{e}_i-[\mathbf{Q}]^T_{j,\star}$;
9 $\quad\boldsymbol{\gamma}:=\frac{1}{(d_j+1)}\big(\mathbf{w}-\frac{1}{C}[\mathbf{S}]_{\star,j}+(\frac{\lambda}{2(d_j+1)}+\frac{1}{C}-1)\mathbf{e}_j\big)$;
10 initialize $\boldsymbol{\xi}_0:=C\cdot\mathbf{e}_j,\quad\boldsymbol{\eta}_0:=\boldsymbol{\gamma},\quad\mathbf{M}_0:=C\cdot\mathbf{e}_j\cdot\boldsymbol{\gamma}^T$;
11 **for** $k=0,1,\cdots,K-1$ **do**
12 $\quad\boldsymbol{\xi}_{k+1}:=C\cdot\mathbf{Q}\cdot\boldsymbol{\xi}_k+C\cdot(\mathbf{v}^T\cdot\boldsymbol{\xi}_k)\cdot\mathbf{u}$;
13 $\quad\boldsymbol{\eta}_{k+1}:=\mathbf{Q}\cdot\boldsymbol{\eta}_k+(\mathbf{v}^T\cdot\boldsymbol{\eta}_k)\cdot\mathbf{u}$;
14 $\quad\mathbf{M}_{k+1}:=\boldsymbol{\xi}_{k+1}\cdot\boldsymbol{\eta}^T_{k+1}+\mathbf{M}_k$;
15 **return** $\tilde{\mathbf{S}}:=\mathbf{S}+\mathbf{M}_K+\mathbf{M}_K^T$;

First, it initializes the transition matrix $\mathbf{Q}$ and in-degree d_j of node j in G (lines 1–2). Using $\mathbf{Q}$ and $\mathbf{S}$, it precomputes the auxiliary vector $\mathbf{w}$ and scalar λ (lines 3–4). Once computed, both $\mathbf{w}$ and λ are memoized for precomputing (i) the vectors $\mathbf{u}$ and $\mathbf{v}$ for a rank-one factorization of $\mathbf{\Delta Q}$, and (ii) the initial vector $\boldsymbol{\gamma}$ for subsequent $\mathbf{M}_k$ iterations (lines 5–9). Then, the algorithm maintains two auxiliary vectors $\boldsymbol{\xi}_k$ and $\boldsymbol{\eta}_k$ to iteratively compute matrix $\mathbf{M}_k$ (lines 10–14). The process continues until the number of iterations reaches a given K. Finally, the new $\tilde{\mathbf{S}}$ is obtained by $\mathbf{M}_K$[16] (line 15).

Example 5 Consider the old digraph G and $\mathbf{S}$ in Fig. 1. When the new edge (i,j) is inserted to G, Inc-uSR computes the new $\tilde{\mathbf{S}}$ as follows, whose results are partially depicted in Column '$\mathsf{sim}_{\mathsf{true}}$' of Fig. 1.

Given the following information from the old $\mathbf{S}$: [17]

$$[\mathbf{S}]_{\star,i}=\big[\,0,\cdots,0,\overset{(f)}{0.246},\overset{(g)}{0},\overset{(h)}{0},\overset{(i)}{0.590},\overset{(j)}{0.310},0,\cdots,0\,\big]^T\in\mathbb{R}^{15\times1},$$

$$[\mathbf{S}]_{\star,j}=\big[\,0,\cdots,0,\overset{(f)}{0.246},\overset{(g)}{0},\overset{(h)}{0},\overset{(i)}{0.310},\overset{(j)}{0.510},0,\cdots,0\,\big]^T\in\mathbb{R}^{15\times1},$$

Inc-uSR first computes $\mathbf{w}$ and λ via lines 3–4:

$$\mathbf{w}=\big[\,\overset{(a)}{0.104},\overset{(b)}{0.139},0,\cdots,0\,\big]^T\in\mathbb{R}^{15\times1},$$

$$\lambda=0.590+\tfrac{1}{0.8}\times0.510-2\times0-\tfrac{1}{0.8}+1=0.978.$$

Since $d_j=2$, the vectors $\mathbf{u}$ and $\mathbf{v}$ for the rank-one decomposition of $\mathbf{\Delta Q}$ can be computed via line 8. Their results are depicted in Example 4.

[16] We can show $\|\mathbf{M}_K-\mathbf{M}\|_{\max}\le C^{K+1}$ with $\mathbf{M}$ in Eq.(25).
[17] Due to space limitations, only the i-th and j-th columns of $\mathbf{S}$ are displayed here, which is sufficient to compute $\tilde{\mathbf{S}}$.

Next, $\boldsymbol{\gamma}$ can be obtained from $\mathbf{w}$ and λ via line 9:

$$\boldsymbol{\gamma} = \tfrac{1}{(2+1)}\big(\mathbf{w} - \tfrac{1}{0.8}[\mathbf{S}]_{\star,j} + (\tfrac{\lambda}{2\times(2+1)} + \tfrac{1}{0.8} - 1)\mathbf{e}_j\big)$$
$$= [\overset{(a)}{0.035}, \overset{(b)}{0.046}, 0, 0, 0, \overset{(f)}{-0.086}\ 0, 0, \overset{(i)}{-0.129}, \overset{(j)}{-0.075}, 0, \cdots, 0]^T \in \mathbb{R}^{15\times 1}$$

In light of $\boldsymbol{\gamma}$, $\mathbf{M}_k$ can be computed via lines 10–14. After $K = 10$ iterations, $\mathbf{M}_K$ can be derived as follows:

	(a)	(b)	(c)	(d)	(e)	(f)	⋯	(i)	(j)	(k)⋯(o)
(a)	−0.005	−0.009	0	0.009					−0.009	
(b)	−0.004	−0.006	0	0.006			0		−0.007	0
(c)	0	0	0	0					0	
(d)	−0.002	−0.002	0	−0.005					0	
⋮ (i)		0					0		0	0
(j)	0.028	0.037	0	0		−0.068		−0.104	−0.060	
⋮ (o)		0					0		0	0

Finally, using $\mathbf{M}_K$ and the old $\mathbf{S}$, the new $\tilde{\mathbf{S}}$ is obtained via line 15, as partly shown in Column '$\mathsf{sim}_{\mathsf{true}}$' of Fig. 1. □

Correctness. Inc-uSR can *correctly* compute new SimRanks for edge update that does not accompany new nodes insertion, as verified by Theorems 1–3.

Complexity. The total complexity of Inc-uSR is bounded by $O(Kn^2)$ time and $O(n^2)$ memory[18]in the worst case for updating *all* similarities of n^2 node-pairs. Precisely, Inc-uSR runs in two phases: preprocessing (lines 1–9), and incremental iterations (lines 10–15):

(a) For the preprocessing, it requires $O(m)$ time in total (m is the number of edges in the old G), which is dominated by computing $\mathbf{w}$ (lines 3), involving the matrix-vector multiplication $\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}$. The time for computing vectors $\mathbf{u}, \mathbf{v}, \boldsymbol{\gamma}$ is bounded by $O(n)$, which includes only vector scaling and additions, *i.e.,* SAXPY.

(b) For the incremental iterative phase, computing $\boldsymbol{\xi}_{k+1}$ and $\boldsymbol{\eta}_{k+1}$ needs $O(m+n)$ time for each iteration (lines 12–13). Computing $\mathbf{M}_{k+1}$ entails $O(n^2)$ time for performing one outer product of two vectors and one matrix addition (lines 14). Thus, the cost of this phase is $O(Kn^2)$ time for K iterations.

Collecting (a) and (b), all n^2 node-pair similarities can be incrementally computed in $O(Kn^2)$ total time, as opposed to the $O(r^4n^2)$ time of its counterpart [12] via incremental SVD.

5 Pruning Unnecessary Node-Pairs in ΔS

After the SimRank update matrix $\boldsymbol{\Delta}\mathbf{S}$ has been characterized as a rank-one Sylvester equation, the pruning techniques in this section can further skip node-pairs with unchanged SimRanks in $\boldsymbol{\Delta}\mathbf{S}$ ("unaffected areas"), to avoid unnecessary recomputation.

[18] In the next sections, we shall substantially reduce its time and memory complexity further.

In practice, we observe that when link updates are small, affected areas in similarity updates $\boldsymbol{\Delta}\mathbf{S}$ are often small as well. As demonstrated in Example 5, many entries in matrix $\mathbf{M}_K$ are 0s, implying that $\boldsymbol{\Delta}\mathbf{S}$ ($= \mathbf{M}_K + \mathbf{M}_K^T$) is a sparse matrix. However, it is a big challenge to identify such "affected areas" in $\boldsymbol{\Delta}\mathbf{S}$ in response to link updates. To address this problem, we first introduce a nice property of the adjacency matrix:

Lemma 1 *Let* $\mathbf{A}$ *be an adjacency matrix. Then* $[\mathbf{A}^k]_{i,j}$ *counts the number of length-*k *paths from node* i *to* j*.*

For example, $[\mathbf{A}^4]_{i,j}$ counts the number of paths $\rho : i \to \circ \to \circ \to \circ \to j$ in G, with $\circ$ denoting any node.

Lemma 1 can be extended to count the number of "specific paths" whose edges are not necessarily in the same direction. For example, we can use $[\mathbf{A}\mathbf{A}^T\mathbf{A}\mathbf{A}^T]_{i,j}$ to count the paths $\rho : i \to \circ \leftarrow \circ \to \circ \leftarrow j$ in G, where $\mathbf{A}$ (*resp.* $\mathbf{A}^T$) appears at the positions 1,3 (*resp.* 2,4), corresponding to the positions of $\to$ (*resp.* $\leftarrow$) in ρ.

As $\mathbf{Q}$ is the row-normalized matrix of $\mathbf{A}^T$, we can prove that $[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{i,j} = 0 \Leftrightarrow [(\mathbf{A}^T)^k\cdot\mathbf{A}^k]_{i,j} = 0$. The following corollary is immediate.

Corollary 1 *Given* $k = 0, 1, \cdots$*, the entry* $[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{i,j}$ *counts the weights of the specific paths whose left* k *edges in "*$\leftarrow$*" direction and right* k *edges in "*$\rightarrow$*" direction:*

$$\underbrace{i \leftarrow \circ \leftarrow \cdots \leftarrow}_{length\ k} \bullet \underbrace{\rightarrow \cdots \rightarrow \circ \rightarrow j}_{length\ k}. \tag{32}$$

Definition 1 We call the paths in Eq.(32) *the symmetric in-link paths of length* $2k$ *for node-pair* (i, j).

By virtue of Eq.(24), the recursive form of SimRank Eq.(1) naturally leads itself to the following series form:

$$[\mathbf{S}]_{a,b} = (1-C)\cdot\sum\nolimits_{k=0}^{\infty} C^k\cdot[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{a,b}. \tag{33}$$

Capitalizing on Corollary 1, Eq.(33) provides a reinterpretation of SimRank: $[\mathbf{S}]_{a,b}$ is the weighted sum of all in-link paths of length $2k$ ($k = 0, 1, 2, \cdots$) for node-pair (a, b). The weight C^k in Eq.(33) is to reduce the contributions of in-link paths with *long* lengths relative to those with *short* ones. The factor $(1 - C)$ aims at normalizing $[\mathbf{S}]_{a,b}$ into $[0, 1]$ since

$$\big\|\sum\nolimits_{k=0}^{\infty} C^k\cdot\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k\big\|_{\max} \le \sum\nolimits_{k=0}^{\infty} C^k \le \tfrac{1}{1-C}.$$

5.1 Affected Areas in ΔS

In light of our interpretation for $\mathbf{S}$ via Eq.(33), we next reinterpret the series $\mathbf{M}$ in Theorem 3, aiming to identify "affected areas" in $\boldsymbol{\Delta}\mathbf{S}$. Due to space limitations,

we mainly focus on the edge insertion case of $d_j > 0$. Other cases have the similar results.

By substituting Eq.(27) back into Eq.(25), we can readily split the series form of $\mathbf{M}$ into three parts:

$$[\mathbf{M}]_{a,b} = \frac{1}{d_j+1}\Big(\underbrace{\sum_{k=0}^{\infty} C^{k+1}\cdot[\tilde{\mathbf{Q}}^k]_{a,j}[\mathbf{S}]_{i,\star}\mathbf{Q}^T\cdot[(\tilde{\mathbf{Q}}^T)^k]_{\star,b}}_{\text{Part 1}} - \underbrace{-\sum_{k=0}^{\infty} C^k[\tilde{\mathbf{Q}}^k]_{a,j}[\mathbf{S}]_{j,\star}[(\tilde{\mathbf{Q}}^T)^k]_{\star,b}}_{\text{Part 2}} + \underbrace{+\mu\sum_{k=0}^{\infty} C^{k+1}[\tilde{\mathbf{Q}}^k]_{a,j}[(\tilde{\mathbf{Q}}^T)^k]_{j,b}}_{\text{Part 3}}\Big)$$

with the scalar $\mu := \frac{\lambda}{2(d_j+1)} + \frac{1}{C} - 1$.

By Lemma 1 and Corollary 1, when edge (i,j) is inserted and $d_j > 0$, Part 1 of $[\mathbf{M}]_{a,b}$ tallies the weighted sum of the following new paths for node-pair (a,b):

$$\overbrace{a \leftarrow \circ\cdots\circ \leftarrow j}^{[\tilde{\mathbf{Q}}^k]_{a,j}} \Leftarrow \overbrace{i \leftarrow \circ\cdots\circ \leftarrow \bullet \rightarrow \circ\cdots\circ \rightarrow \star}^{[\mathbf{S}]_{i,\star}} \overbrace{\rightarrow}^{\mathbf{Q}^T} \overbrace{\blacktriangle \rightarrow \cdots\circ \rightarrow b}^{[(\tilde{\mathbf{Q}}^T)^k]_{\blacktriangle,b}} \quad (34)$$

(length k; all symmetric in-link paths for node-pair $(i,\star)$; length k)

Such paths are the concatenation of four types of sub-paths (as depicted above) associated with four matrices, respectively, $[\tilde{\mathbf{Q}}^k]_{a,j}, [\mathbf{S}]_{i,\star}, \mathbf{Q}^T, [(\tilde{\mathbf{Q}}^T)^k]_{\blacktriangle,b}$, plus the inserted edge $j \Leftarrow i$. When such entire concatenated paths exist in the new graph, they should be accommodated for assessing the new SimRank $[\tilde{\mathbf{S}}]_{a,b}$ in response to the edge insertion (i,j) because our reinterpretation of SimRank indicates that SimRank counts *all* the symmetric in-link paths, and the entire concatenated paths can prove to be symmetric in-link paths.

Likewise, Parts 2 and 3 of $[\mathbf{M}]_{a,b}$, respectively, tally the weighted sum of the following paths for pair (a,b):

$$\overbrace{a \leftarrow \circ\cdots\circ \leftarrow}^{[\tilde{\mathbf{Q}}^k]_{a,j}} j \overbrace{\leftarrow \circ\cdots\circ \leftarrow \bullet \rightarrow \circ\cdots\circ \rightarrow}^{[\mathbf{S}]_{j,\star}} \star \overbrace{\rightarrow \cdots\circ \rightarrow b}^{[(\tilde{\mathbf{Q}}^T)^k]_{\star,b}} \quad (35)$$

(length k; all symmetric in-link paths for $(j,\star)$; length k)

$$\overbrace{a \leftarrow \circ\cdots\circ \leftarrow}^{[\tilde{\mathbf{Q}}^k]_{a,j}} j \overbrace{\rightarrow \circ\cdots\circ \rightarrow b}^{[(\tilde{\mathbf{Q}}^T)^k]_{j,b}} \quad (36)$$

(length k; length k)

Indeed, when edge (i,j) is inserted, only these three kinds of paths have extra contributions for $\mathbf{M}$ (therefore for $\mathbf{\Delta S}$). As incremental updates in SimRank merely tally these paths, node-pairs without having such paths could be safely pruned. In other words, for those pruned node-pairs, the three kinds of paths will have "zero contributions" to the changes in $\mathbf{M}$ in response to edge insertion. Thus, after pruning, the remaining node-pairs in G constitute the "affected areas" of $\mathbf{M}$.

We next identify "affected areas" of $\mathbf{M}$, by pruning redundant node-pairs in G, based on the following.

Theorem 5 *For the edge (i,j) insertion, let $\mathcal{O}(a)$ and $\tilde{\mathcal{O}}(a)$ be the out-neighbors of node a in old G and new $G \cup \{(i,j)\}$, respectively. Let $\mathbf{M}_k$ be the k-th iterative matrix in Line 14 of Algorithm 1, and let*

$$\mathcal{F}_1 := \{b \mid b \in \mathcal{O}(y),\ \exists y,\ s.t.\ [\mathbf{S}]_{i,y} \neq 0\} \quad (37)$$

$$\mathcal{F}_2 := \begin{cases} \varnothing & (d_j = 0) \\ \{y \mid [\mathbf{S}]_{j,y} \neq 0\} & (d_j > 0) \end{cases} \quad (38)$$

$$\mathcal{A}_k \times \mathcal{B}_k := \quad (39)$$

$$\begin{cases} \{j\} \times (\mathcal{F}_1 \cup \mathcal{F}_2 \cup \{j\}) & (k=0) \\ \{(a,b) \mid a \in \tilde{\mathcal{O}}(x),\ b \in \tilde{\mathcal{O}}(y),\ \exists x,\ \exists y,\ s.t.\ [\mathbf{M}_{k-1}]_{x,y} \neq 0\} & (k>0) \end{cases}$$

Then, for every iteration $k = 0, 1, \cdots$, the matrix $\mathbf{M}_k$ has the following sparse property:

$$[\mathbf{M}_k]_{a,b} = 0 \quad \textit{for all } (a,b) \notin (\mathcal{A}_k \times \mathcal{B}_k) \cup (\mathcal{A}_0 \times \mathcal{B}_0).$$

For the edge (i,j) deletion case, all the above results hold except that, in Eq.(38), the conditions $d_j = 0$ and $d_j > 0$ are, respectively, replaced by $d_j = 1$ and $d_j > 1$.

Proof We only show the edge insertion case $d_j > 0$, due to space limits. The proofs of other cases are similar.

For $k = 0$, it follows from Eq.(25) that $[\mathbf{M}_0]_{a,b} = [\mathbf{e}_j]_a[\boldsymbol{\gamma}]_b$. Thus, $\forall (a,b) \notin \mathcal{A}_0 \times \mathcal{B}_0$, there are two cases: (i) $a \neq j$, or (ii) $a = j$, $b \in {\mathcal{F}_1}^C \cap {\mathcal{F}_2}^C$, and $b \neq j$.

For case (i), $[\mathbf{e}_j]_a = 0$ for $a \neq j$. Thus, $[\mathbf{M}_0]_{a,b} = 0$. For case (ii), $[\mathbf{e}_j]_a = 1$ for $a = j$. Thus, $[\mathbf{M}_0]_{a,b} = [\boldsymbol{\gamma}]_b$, where $[\boldsymbol{\gamma}]_b$ is the linear combinations of the 3 terms: $[\mathbf{Q}]_{b,\star}\cdot[\mathbf{S}]_{\star,i}$, $[\mathbf{S}]_{b,j}$, and $[\mathbf{e}_j]_b$, according to the case of $d_j > 0$ in Eq.(27).

Next, our goal is to show the 3 terms are all 0s. (a) For $b \notin \mathcal{F}_1$, by definition in Eq.(37), $b \in \mathcal{O}(y)$ for $\forall y$, we have $[\mathbf{S}]_{i,y} = 0$. Due to symmetry, $b \in \mathcal{O}(y) \Leftrightarrow y \in \mathcal{I}(b)$, which implies that $[\mathbf{S}]_{i,y} = 0$ for $\forall y \in \mathcal{I}(b)$. [19] Thus, $[\mathbf{Q}]_{b,\star}\cdot[\mathbf{S}]_{\star,i} = \frac{1}{\mathcal{I}(b)}\sum_{x\in\mathcal{I}(b)}[\mathbf{S}]_{x,i} = 0$. (b) For $b \notin \mathcal{F}_2$, it follows from the case $d_j > 0$ in Eq.(38) that $[\mathbf{S}]_{j,b} = 0$. Hence, by $\mathbf{S}$ symmetry, $[\mathbf{S}]_{b,j} = [\mathbf{S}]_{j,b} = 0$. (c) $[\mathbf{e}_j]_b = 0$ since $b \neq j$.

Taking (a)–(c) together, it follows that $[\mathbf{M}_0]_{a,b} = 0$, which completes the proof for the case $k = 0$.

For $k > 0$, one can readily prove that the k-th iterative $\mathbf{M}_k$ in Line 14 of Algorithm 1 is the first k-th partial sum of $\mathbf{M}$ in Eq.(25). Thus, $\mathbf{M}_{k+1}$ can be derived from $\mathbf{M}_k$ as follows:

$$\mathbf{M}_k = C\cdot\tilde{\mathbf{Q}}\cdot\mathbf{M}_{k-1}\cdot\tilde{\mathbf{Q}}^T + C\cdot\mathbf{e}_j\cdot\boldsymbol{\gamma}^T.$$

Thus, the (a,b)-entry form of the above equation is

$$[\mathbf{M}_k]_{a,b} = \frac{C}{|\tilde{\mathcal{I}}(a)||\tilde{\mathcal{I}}(b)|}\sum_{x\in\tilde{\mathcal{I}}(a)}\sum_{y\in\tilde{\mathcal{I}}(b)}[\mathbf{M}_{k-1}]_{x,y} + C\cdot[\mathbf{e}_j]_a\cdot[\boldsymbol{\gamma}]_b.$$

[19] Herein, we denote by $\mathcal{I}(a)$ the in-neighbor set of node a.

To show that $[\mathbf{M}_k]_{a,b} = 0$ for $(a,b) \notin \mathcal{A}_0 \times \mathcal{B}_0 \cup \mathcal{A}_k \times \mathcal{B}_k$, we follow the 2 steps: (i) For $(a,b) \notin \mathcal{A}_0 \times \mathcal{B}_0$, as proved in the case $k = 0$, the term $C \cdot [\mathbf{e}_j]_a [\boldsymbol{\gamma}]_b$ in the above equation is obviously 0. (ii) For $(a,b) \notin \mathcal{A}_k \times \mathcal{B}_k$, by virtue of Eq.(39), $a \in \tilde{\mathcal{O}}(x), b \in \tilde{\mathcal{O}}(y)$, for $\forall x, y$, we have $[\mathbf{M}_{k-1}]_{x,y} = 0$. Hence, by symmetry, it follows that $x \in \tilde{\mathcal{I}}(a), y \in \tilde{\mathcal{I}}(b)$, $[\mathbf{M}_{k-1}]_{x,y} = 0$.

Taking (i) and (ii) together, we can conclude that $[\mathbf{M}_k]_{a,b} = 0$ for $(a,b) \notin \mathcal{A}_0 \times \mathcal{B}_0 \cup \mathcal{A}_k \times \mathcal{B}_k$. □

Theorem 5 provides a pruning strategy to iteratively eliminate node-pairs with a-priori zero values in $\mathbf{M}_k$ (thus in $\mathbf{\Delta S}$). Hence, by Theorem 5, when edge (i,j) is updated, we just need to consider node-pairs in $(\mathcal{A}_k \times \mathcal{B}_k) \cup (\mathcal{A}_0 \times \mathcal{B}_0)$ for incrementally updating $\mathbf{\Delta S}$.

Intuitively, $\mathcal{F}_1$ in Eq.(37) captures the nodes "▲" in (34). To be specific, $\mathcal{F}_1$ can be obtained via 2 phases: (i) For the given node i, we first build an intermediate set $\mathcal{T} := \{y | [\mathbf{S}]_{i,y} \neq 0\}$, which consists of nodes "⋆" in (34). (ii) For each node $x \in \mathcal{T}$, we then find all out-neighbors of x in G, which produces $\mathcal{F}_1$, *i.e.,* $\mathcal{F}_1 = \bigcup_{x \in \mathcal{T}} \mathcal{O}(x)$. Analogously, the set $\mathcal{F}_2$ in Eq.(38), in the case of $d_j > 0$, consists of the nodes "⋆" depicted in (35). When $d_j = 0$, $\mathcal{F}_2 = \varnothing$ as the term $[\mathbf{S}]_{\star,i}$ is not in the expression of $\boldsymbol{\gamma}$ in Eq.(26), in contrast to the case $d_j > 0$.

After obtaining $\mathcal{F}_1$ and $\mathcal{F}_2$, we can readily find $\mathcal{A}_0 \times \mathcal{B}_0$, according to Eq.(39). For $k > 0$, to iteratively derive the node-pair set $\mathcal{A}_k \times \mathcal{B}_k$, we take the following two steps: (i) we first construct a node-pair set $\mathcal{T}_1 \times \mathcal{T}_2 := \{(x,y) | [\mathbf{M}_{k-1}]_{x,y} \neq 0\}$. (ii) For every node $x \in \mathcal{T}_1$ (*resp.* $y \in \mathcal{T}_2$), we then find all out-neighbors of x (*resp.* y) in $G \cup \{(i,j)\}$, which yields $\mathcal{A}_k$ (*resp.* $\mathcal{B}_k$), *i.e.,* $\mathcal{A}_k = \bigcup_{x \in \mathcal{T}_1} \tilde{\mathcal{O}}(x)$ and $\mathcal{B}_k = \bigcup_{y \in \mathcal{T}_2} \tilde{\mathcal{O}}(y)$.

The node selectivity of Theorem 5 hinges on $\mathbf{\Delta S}$ sparsity. Since real graphs are constantly updated with *minor* changes, $\mathbf{\Delta S}$ is often *sparse* in general. Hence, many node-pairs with zero scores in $\mathbf{\Delta S}$ can be discarded. As demonstrated by our experiments in Fig.10, 76.3% paper-pairs on DBLP can be pruned, significantly reducing unnecessary similarity recomputations.

Example 6 Recall Example 5 and the old graph G in Fig. 1. When edge (i,j) is inserted to G, according to Theorem 5, $\mathcal{F}_1 = \{a,b\}$, $\mathcal{F}_2 = \{f,i,j\}$, $\mathcal{A}_0 \times \mathcal{B}_0 = \{j\} \times \{a,b,f,i,j\}$. Hence, instead of computing the entire vector $\boldsymbol{\gamma}$ in Eqs.(26) and (27), we only need to compute part of its entries $[\boldsymbol{\gamma}]_x$ for $\forall x \in \mathcal{B}_0$.

For the first iteration, since $\mathcal{A}_1 \times \mathcal{B}_1 = \{a,b\} \times \{a,b,d,j\}$, then we only need to compute 18 $(= 3 \times 6)$ entries $[\mathbf{M}_1]_{x,y}$ for $\forall (x,y) \in \{a,b,j\} \times \{a,b,d,f,i,j\}$, skipping the computations of 207 $(= 15^2 - 18)$ remaining entries in $\mathbf{M}_1$. After $K = 10$ iterations, many unnecessary node-pairs are pruned, as in part highlighted in the gray rows of the table in Fig. 1. □

Algorithm 2: Inc-SR $(G, \mathbf{S}, K, (i,j), C)$

Input / Output: the same as Algorithm 1.
1-2 the same as Algorithm 1 ;
3 find $\mathcal{B}_0$ via Eq.(39) ;
memoize $[\mathbf{w}]_b := [\mathbf{Q}]_{b,\star} \cdot [\mathbf{S}]_{\star,i}$, for all $b \in \mathcal{B}_0$;
4-12 almost the same as Algorithm 1 except that the computations of the entire vector $\boldsymbol{\gamma}$ in Lines 6, 8, 10, 12 are replaced by the computations of only parts of entries in $\boldsymbol{\gamma}$, respectively, *e.g.,* in Line 6 of Algorithm 1, "$\boldsymbol{\gamma} := \mathbf{w} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j$" are replaced by "$[\boldsymbol{\gamma}]_b := [\mathbf{w}]_b + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot [\mathbf{e}_j]_b$, for all $b \in \mathcal{B}_0$" ;
13 $[\boldsymbol{\xi}_0]_j := C$, $[\boldsymbol{\eta}_0]_b := [\boldsymbol{\gamma}]_b$, $[\mathbf{M}_0]_{j,b} := C \cdot [\boldsymbol{\gamma}]_b, \forall b \in \mathcal{B}_0$;
14 **for** $k = 1, \cdots, K$ **do**
15 find $\mathcal{A}_k \times \mathcal{B}_k$ via Eq.(39) ;
16 memoize $\sigma_1 := C \cdot (\mathbf{v}^T \cdot \boldsymbol{\xi}_{k-1})$, $\sigma_2 := \mathbf{v}^T \cdot \boldsymbol{\eta}_{k-1}$;
17 $[\boldsymbol{\xi}_k]_a := C \cdot [\mathbf{Q}]_{a,\star} \cdot \boldsymbol{\xi}_{k-1} + \sigma_1 \cdot [\mathbf{u}]_a$, for all $a \in \mathcal{A}_k$;
18 $[\boldsymbol{\eta}_k]_b := [\mathbf{Q}]_{b,\star} \cdot \boldsymbol{\eta}_{k-1} + \sigma_2 \cdot [\mathbf{u}]_b$, for all $b \in \mathcal{B}_k$;
19 $[\mathbf{M}_k]_{a,b} := [\boldsymbol{\xi}_k]_a \cdot [\boldsymbol{\eta}_k]_b + [\mathbf{M}_{k-1}]_{a,b}$, $\forall (a,b) \in \mathcal{A}_k \times \mathcal{B}_k$;
20 $[\tilde{\mathbf{S}}]_{a,b} := [\mathbf{S}]_{a,b} + [\mathbf{M}_K]_{a,b} + [\mathbf{M}_K]_{b,a}$, $\forall (a,b) \in \mathcal{A}_K \times \mathcal{B}_K$;
21 **return** $\tilde{\mathbf{S}}$;

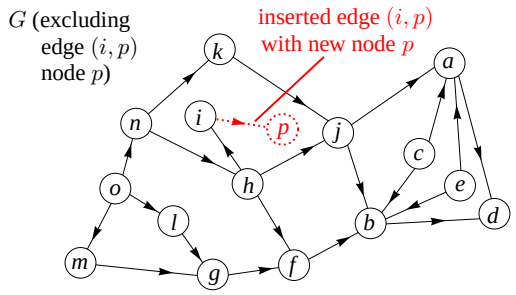
5.2 Algorithm

We provide a complete incremental algorithm for computing SimRank, referred to as Inc-SR (in Algorithm 2), by incorporating our pruning strategy into Inc-uSR.

Correctness. The algorithm Inc-SR can *correctly* prune the node-pairs with a-priori zero scores in $\mathbf{\Delta S}$, which is verified by Theorem 5. It also *correctly* returns the new similarities, as evidenced by Theorems 1–3.

Complexity. The total time of Inc-SR is $O(K(nd + |\mathsf{AFF}|))$ for K iterations, where d is the average in-degree of G, and $|\mathsf{AFF}| := \text{avg}_{k \in [0,K]}(|\mathcal{A}_k| \cdot |\mathcal{B}_k|)$ with $\mathcal{A}_k, \mathcal{B}_k$ in Eq.(39), being the average size of "affected areas" in $\mathbf{M}_k$ for K iterations. More concretely, (a) for the preprocessing, finding $\mathcal{B}_0$ (line 3) needs $O(dn)$ time. Utilizing $\mathcal{B}_0$, computing $[\mathbf{w}]_b$ reduces from $O(m)$ to $O(d|\mathcal{B}_0|)$ time, with $|\mathcal{B}_0| \ll n$. Analogously, $\boldsymbol{\gamma}$ in lines 6,8,10,12 of Algorithm 1 needs only $O(|\mathcal{B}_0|)$ time. (b) For each iteration, finding $\mathcal{A}_k \times \mathcal{B}_k$ (line 15) entails $O(dn)$ time. Memoizing σ_1, σ_2 needs $O(n)$ time (line 16). Computing $\boldsymbol{\xi}$ (*resp.* $\boldsymbol{\eta}$) reduces from $O(m)$ to $O(d|\mathcal{A}_k|)$ (*resp.* $O(d|\mathcal{B}_k|)$) time (lines 17–18). Computing $[\mathbf{M}_k]_{a,b}$ reduces from $O(n^2)$ to $O(|\mathcal{A}_k||\mathcal{B}_k|)$ time (line 19). Thus, the total time complexity can be bounded by $O(K(nd + |\mathsf{AFF}|))$ for K iterations.

It is worth mentioning that Inc-SR, in the worst case, has the same complexity bound of Inc-uSR. However, in practice, $|\mathsf{AFF}| \ll n^2$, as demonstrated by our experimental study in Fig.11, since real graphs are constantly updated with *small* changes. Hence, $O(K(nd + |\mathsf{AFF}|))$ is generally much smaller than $O(Kn^2)$. In the next section, we shall further confirm the efficiency of Inc-SR by conducting extensive experiments.



Node-Pair	in G sim_{old}	in $G \cup \Delta G$ sim_{new}
(a,b)	**0.0745**	**0.0745**
(a,p)	—	0.0828
(b,p)	—	0.1114
(f,i)	**0.2464**	**0.2464**
(f,j)	**0.2064**	**0.2064**
(g,h)	**0.128**	**0.128**
(g,k)	**0.128**	**0.128**
(h,k)	**0.288**	**0.288**
(i,j)	**0.3104**	**0.3104**
(l,m)	**0.16**	**0.16**
(l,n)	**0.16**	**0.16**
(m,n)	**0.16**	**0.16**

Fig. 3: Incrementally update SimRank when a new edge (i,p) with $i \in V$ and $p \notin V$ is inserted into $G=(V,E)$

6 Edge Update with Nodes Insertion

In this section, we focus on the edge update that accompanies new node(s) insertion. Specifically, given a new edge (i,j) to be inserted into the old graph $G=(V,E)$, we consider the following cases when

(C1) $i \in V$ and $j \notin V$; (in Subsection 6.1)
(C2) $i \notin V$ and $j \in V$; (in Subsection 6.2)
(C3) $i \notin V$ and $j \notin V$. (in Subsection 6.3)

For each case, we devise an efficient incremental algorithm that can support new nodes insertion and can accurately update only "affected areas" of SimRanks.

Remark 3 Let $n=|V|$, without loss of generality, it can be tacitly assumed that
a) in case (C1), new node $j \notin V$ is indexed by $(n+1)$;
b) in case (C2), new node $i \notin V$ is indexed by $(n+1)$;
c) in case (C3), new nodes $i \notin V$ and $j \notin V$ are indexed by $(n+1)$ and $(n+2)$, respectively.

6.1 Inserting an edge (i,j) with $i \in V$ and $j \notin V$

In this case, the inserted new edge (i,j) accompanies the insertion of a new node j. Thus, the size of the new SimRank matrix $\tilde{\mathbf{S}}$ is different from that of the old $\mathbf{S}$. As a result, we cannot simply evaluate the changes to $\mathbf{S}$ by adopting $\tilde{\mathbf{S}}-\mathbf{S}$ as we did in Section 4.

To resolve this problem, we introduce the block matrix representation of new matrices for edge insertion. Firstly, when a new edge $(i,j)_{i\in V, j\notin V}$ is inserted to G, the new transition matrix $\tilde{\mathbf{Q}}$ can be described as

$$\tilde{\mathbf{Q}} = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{e}_i^T & 0 \end{array}\right] \begin{array}{l} \} \ n \text{ rows} \\ \rightarrow \text{row } j \end{array} \in \mathbb{R}^{(n+1)\times(n+1)} \tag{40}$$

Intuitively, $\tilde{\mathbf{Q}}$ is formed by bordering the old $\mathbf{Q}$ by 0s except $[\tilde{\mathbf{Q}}]_{j,i}=1$. Utilizing this block structure of $\tilde{\mathbf{Q}}$, we can obtain the new SimRank matrix, which exhibits a similar block structure, as shown below:

Theorem 6 *Given an old digraph $G=(V,E)$, if there is a new edge (i,j) with $i \in V$ and $j \notin V$ to be inserted, then the new SimRank matrix becomes*

$$\tilde{\mathbf{S}} = \left[\begin{array}{c|c} \mathbf{S} & \mathbf{y} \\ \hline \mathbf{y}^T & C[\mathbf{S}]_{i,i}+(1-C) \end{array}\right] \begin{array}{l} \} \ n \text{ rows} \\ \rightarrow \text{row } j \end{array} \quad \text{with } \mathbf{y} = C\mathbf{Q}[\mathbf{S}]_{\star,i} \tag{41}$$

where $\mathbf{S} \in \mathbb{R}^{n\times n}$ is the old SimRank matrix of G.

Proof We substitute the new $\tilde{\mathbf{Q}}$ in Eq.(40) back into the SimRank equation $\tilde{\mathbf{S}} = C\cdot\tilde{\mathbf{Q}}\cdot\tilde{\mathbf{S}}\cdot\tilde{\mathbf{Q}}^T+(1-C)\cdot\mathbf{I}_{n+1}$:

$$\mathbf{S} := \left[\begin{array}{c|c} \tilde{\mathbf{S}}_{11} & \tilde{\mathbf{S}}_{12} \\ \hline \tilde{\mathbf{S}}_{21} & \tilde{\mathbf{S}}_{22} \end{array}\right] = C \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{e}_i^T & 0 \end{array}\right] \left[\begin{array}{c|c} \tilde{\mathbf{S}}_{11} & \tilde{\mathbf{S}}_{12} \\ \hline \tilde{\mathbf{S}}_{21} & \tilde{\mathbf{S}}_{22} \end{array}\right] \left[\begin{array}{c|c} \mathbf{Q}^T & \mathbf{e}_i \\ \hline \mathbf{0} & 0 \end{array}\right] + (1-C) \left[\begin{array}{c|c} \mathbf{I}_n & \mathbf{0} \\ \hline \mathbf{0} & 1 \end{array}\right]$$

By expanding the right-hand side, we can obtain

$$\left[\begin{array}{c|c} \tilde{\mathbf{S}}_{11} & \tilde{\mathbf{S}}_{12} \\ \hline \tilde{\mathbf{S}}_{21} & \tilde{\mathbf{S}}_{22} \end{array}\right] = \left[\begin{array}{c|c} C\mathbf{Q}\tilde{\mathbf{S}}_{11}\mathbf{Q}^T+(1-C)\mathbf{I}_n & C\mathbf{Q}\tilde{\mathbf{S}}_{11}\mathbf{e}_i \\ \hline C\mathbf{e}_i{}^T\tilde{\mathbf{S}}_{11}\mathbf{Q}^T & C\mathbf{e}_i{}^T\tilde{\mathbf{S}}_{11}\mathbf{e}_i+(1-C) \end{array}\right]$$

The above block matrix equation implies that

$$\tilde{\mathbf{S}}_{11} = C\mathbf{Q}\tilde{\mathbf{S}}_{11}\mathbf{Q}^T+(1-C)\mathbf{I}_n$$

Due to the uniqueness of $\mathbf{S}$ in Eq.(1), it follows that

$$\tilde{\mathbf{S}}_{11} = \mathbf{S}$$

Thus, we have

$$\tilde{\mathbf{S}}_{12} = \tilde{\mathbf{S}}_{21}^T = C\mathbf{Q}\tilde{\mathbf{S}}_{11}\mathbf{e}_i = C\mathbf{Q}[\mathbf{S}]_{\star,i}$$
$$\tilde{\mathbf{S}}_{22} = C\mathbf{e}_i{}^T\tilde{\mathbf{S}}_{11}\mathbf{e}_i+(1-C) = C[\mathbf{S}]_{i,i}+(1-C)$$

Combining all blocks of $\tilde{\mathbf{S}}$ together yields Eq.(41). □

Theorem 6 provides an efficient incremental way of computing the new SimRank matrix $\tilde{\mathbf{S}}$ for unit insertion of the case (C1). Precisely, the new $\tilde{\mathbf{S}}$ is formed by bordering the old $\mathbf{S}$ by the auxiliary vector $\mathbf{y}$. To obtain $\mathbf{y}$ (and thereby $\tilde{\mathbf{S}}$), we just need use the i-th column of $\mathbf{S}$ with one matrix-vector multiplication ($\mathbf{Q}[\mathbf{S}]_{\star,i}$). Thus, the total cost of computing new $\tilde{\mathbf{S}}$ requires $O(m)$ time, as illustrated in Algorithm 3.

Algorithm 3: Inc-uSR-C1 $(G,(i,j),\mathbf{S},C)$

Input : a directed graph $G=(V,E)$,
a new edge $(i,j)_{i\in V,\ j\notin V}$ inserted to G,
the old similarities $\mathbf{S}$ in G,
the damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$.

1 initialize the transition matrix $\mathbf{Q}$ in G ;
2 compute $\mathbf{y}:=C\cdot\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}$;
3 compute $z:=C\cdot[\mathbf{S}]_{i,i}+(1-C)$;
4 **return** $\tilde{\mathbf{S}}:=\left[\begin{array}{c|c}\mathbf{S}&\mathbf{y}\\\hline\mathbf{y}^T&z\end{array}\right]$;

Example 7 Consider the citation digraph G in Fig. 3. If the new edge (i,p) with new node p is inserted to G, the new $\tilde{\mathbf{S}}$ can be updated from the old $\mathbf{S}$ as follows:

According to Theorem 6, since $C=0.8$ and

$$[\mathbf{S}]_{\star,i}=[\overset{(a)}{0},\ \cdots,\ \overset{(e)}{0},\ \overset{(f)}{0.2464},\ \overset{(g)}{0},\ \overset{(h)}{0},\ \overset{(i)}{0.5904},\ \overset{(j)}{0.3104},\ \overset{(k)}{0},\ \cdots,\ \overset{(o)}{0}\ ]^T$$

it follows that

$$\tilde{\mathbf{S}}=\left[\begin{array}{c|c}\mathbf{S}&\mathbf{y}\\\hline\mathbf{y}^T&z\end{array}\right]\quad\text{with } z=0.8[\mathbf{S}]_{i,i}+(1-0.8)=0.6723$$

$$\mathbf{y}=0.8\mathbf{Q}[\mathbf{S}]_{\star,i}=[\overset{(a)}{0.0828},\ \overset{(b)}{0.1114},\ \overset{(c)}{0},\ \cdots,\ \overset{(o)}{0}\ ]^T\in\mathbb{R}^{15\times1}\ \square$$

6.2 Inserting an edge (i,j) with $i\notin V$ and $j\in V$

We now focus on the case (C2), the insertion of an edge (i,j) with $i\notin V$ and $j\in V$. Similar to the case (C1), the new edge accompanies the insertion of a new node i. Hence, $\tilde{\mathbf{S}}-\mathbf{S}$ makes no sense.

However, in this case, the dynamic computation of SimRank is far more complicated than that of the case (C1), in that such an edge insertion not only increases the dimension of the old transition matrix $\mathbf{Q}$ by one, but also changes several original elements of $\mathbf{Q}$, which may recursively influence SimRank similarities. Specifically, the following theorem shows, in the case (C2), how $\mathbf{Q}$ changes with the insertion of an edge $(i,j)_{i\notin V,j\in V}$.

Theorem 7 *Given an old digraph $G=(V,E)$, if there is a new edge (i,j) with $i\notin V$ and $j\in V$ to be added to G, then the new transition matrix can be expressed as*

$$\tilde{\mathbf{Q}}=\left[\begin{array}{c|c}\hat{\mathbf{Q}}&\frac{1}{d_j+1}\mathbf{e}_j\\\hline\mathbf{0}&0\end{array}\right]\ \begin{array}{l}\}\ n\ rows\\\rightarrow row\ i\end{array}\quad with\ \hat{\mathbf{Q}}:=\mathbf{Q}-\tfrac{1}{d_j+1}\mathbf{e}_j[\mathbf{Q}]_{j,\star}\qquad(42)$$

where $\mathbf{Q}$ is the old transition matrix of G.

Proof When edge (i,j) with $i\notin V$ and $j\in V$ is added, there will be two changes to the old $\mathbf{Q}$:
(i) All nonzeros in $[\mathbf{Q}]_{j,\star}$ are updated from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$:

$$[\hat{\mathbf{Q}}]_{j,\star}=\tfrac{d_j}{d_j+1}[\mathbf{Q}]_{j,\star}=[\mathbf{Q}]_{j,\star}-\tfrac{1}{d_j+1}[\mathbf{Q}]_{j,\star}\qquad(43)$$

(ii) The size of the old $\tilde{\mathbf{Q}}$ is added by 1, with new entry $[\mathbf{Q}]_{j,i}=\frac{1}{d_j+1}$ in the bordered areas and 0s elsewhere:

$$\tilde{\mathbf{Q}}=\left[\begin{array}{c|c}\hat{\mathbf{Q}}&\frac{1}{d_j+1}\mathbf{e}_j\\\hline\mathbf{0}&0\end{array}\right]\qquad(44)$$

Combining Eqs.(43) and (44) yields (42). $\square$

Theorem 7 exhibits a special structure of the new $\tilde{\mathbf{Q}}$: it is formed by bordering $\hat{\mathbf{Q}}$ by 0s except $[\tilde{\mathbf{Q}}]_{j,i}=\frac{1}{d_j+1}$, where $\hat{\mathbf{Q}}$ is a rank-one update of the old $\mathbf{Q}$. The block structure of $\tilde{\mathbf{Q}}$ inspires us to partition the new SimRank matrix $\tilde{\mathbf{S}}$ conformably into the similar block structure:

$$\tilde{\mathbf{S}}=\left[\begin{array}{c|c}\tilde{\mathbf{S}}_{\mathbf{11}}&\tilde{\mathbf{S}}_{\mathbf{12}}\\\hline\tilde{\mathbf{S}}_{\mathbf{21}}&\tilde{\mathbf{S}}_{\mathbf{22}}\end{array}\right]\quad\text{where}\quad\begin{array}{l}\tilde{\mathbf{S}}_{\mathbf{11}}\in\mathbb{R}^{n\times n},\ \tilde{\mathbf{S}}_{\mathbf{12}}\in\mathbb{R}^{n\times1},\\\tilde{\mathbf{S}}_{\mathbf{21}}\in\mathbb{R}^{1\times n},\ \tilde{\mathbf{S}}_{\mathbf{22}}\in\mathbb{R}.\end{array}$$

To determine each block of $\tilde{\mathbf{S}}$ with respect to the old $\mathbf{S}$, we next present the following theorem.

Theorem 8 *If there is a new edge (i,j) with $i\notin V$ and $j\in V$ to be added to the old digraph $G=(V,E)$, then there exists a vector*

$$\mathbf{z}=\alpha\mathbf{e}_j-\mathbf{y}\ \textit{with}\ \mathbf{y}:=\mathbf{Q}\mathbf{S}[\mathbf{Q}]_{j,\star}^T\ \textit{and}\ \alpha:=\tfrac{\mathbf{y}_j+1-C}{2(d_j+1)}\qquad(45)$$

such that the new SimRank matrix $\tilde{\mathbf{S}}$ is expressible as

$$\tilde{\mathbf{S}}=\left[\begin{array}{c|c}\mathbf{S}+\mathbf{\Delta}\tilde{\mathbf{S}}_{\mathbf{11}}&\mathbf{0}\\\hline\mathbf{0}&1-C\end{array}\right]\ \begin{array}{l}\}\ n\ rows\\\rightarrow row\ i\end{array}\qquad(46)$$

where $\mathbf{S}$ is the old SimRank of G, and $\mathbf{\Delta}\tilde{\mathbf{S}}_{\mathbf{11}}$ satisfies the rank-two Sylvester equation:

$$\mathbf{\Delta}\tilde{\mathbf{S}}_{\mathbf{11}}=C\hat{\mathbf{Q}}\mathbf{\Delta}\tilde{\mathbf{S}}_{\mathbf{11}}\hat{\mathbf{Q}}^T+\tfrac{C}{d_j+1}\left(\mathbf{e}_j\mathbf{z}^T+\mathbf{z}{\mathbf{e}_j}^T\right)\qquad(47)$$

with $\hat{\mathbf{Q}}$ being defined by Theorem 7.

Proof We plug $\tilde{\mathbf{Q}}$ of Eq.(42) into the SimRank formula:

$$\tilde{\mathbf{S}}=C\cdot\tilde{\mathbf{Q}}\cdot\tilde{\mathbf{S}}\cdot\tilde{\mathbf{Q}}^T+(1-C)\cdot\mathbf{I}_{n+1},$$

which produces

$$\tilde{\mathbf{S}}=\left[\begin{array}{c|c}\tilde{\mathbf{S}}_{\mathbf{11}}&\tilde{\mathbf{S}}_{\mathbf{12}}\\\hline\tilde{\mathbf{S}}_{\mathbf{21}}&\tilde{\mathbf{S}}_{\mathbf{22}}\end{array}\right]=C\left[\begin{array}{c|c}\hat{\mathbf{Q}}&\frac{1}{d_j+1}\mathbf{e}_j\\\hline\mathbf{0}&0\end{array}\right]\left[\begin{array}{c|c}\tilde{\mathbf{S}}_{\mathbf{11}}&\tilde{\mathbf{S}}_{\mathbf{12}}\\\hline\tilde{\mathbf{S}}_{\mathbf{21}}&\tilde{\mathbf{S}}_{\mathbf{22}}\end{array}\right]\left[\begin{array}{c|c}\hat{\mathbf{Q}}^T&\mathbf{0}\\\hline\frac{1}{d_j+1}\mathbf{e}_j^T&0\end{array}\right]+(1-C)\left[\begin{array}{c|c}\mathbf{I}_n&\mathbf{0}\\\hline\mathbf{0}&1\end{array}\right]$$

By using block matrix multiplications, the above equation can be simplified as

$$\left[\begin{array}{c|c}\tilde{\mathbf{S}}_{\mathbf{11}}&\tilde{\mathbf{S}}_{\mathbf{12}}\\\hline\tilde{\mathbf{S}}_{\mathbf{21}}&\tilde{\mathbf{S}}_{\mathbf{22}}\end{array}\right]=C\left[\begin{array}{c|c}\mathbf{P}&\mathbf{0}\\\hline\mathbf{0}&0\end{array}\right]+(1-C)\left[\begin{array}{c|c}\mathbf{I}_n&\mathbf{0}\\\hline\mathbf{0}&1\end{array}\right]\qquad(48)$$

$$\begin{aligned}\text{with }\mathbf{P}=&\hat{\mathbf{Q}}\tilde{\mathbf{S}}_{\mathbf{11}}\hat{\mathbf{Q}}^T+\tfrac{1}{(d_j+1)^2}\mathbf{e}_j\tilde{\mathbf{S}}_{\mathbf{22}}{\mathbf{e}_j}^T\\&+\tfrac{1}{d_j+1}\mathbf{e}_j\tilde{\mathbf{S}}_{\mathbf{21}}\hat{\mathbf{Q}}^T+\tfrac{1}{d_j+1}\hat{\mathbf{Q}}\tilde{\mathbf{S}}_{\mathbf{12}}{\mathbf{e}_j}^T\end{aligned}\qquad(49)$$

Block-wise comparison of both sides of Eq.(48) yields

$$\begin{cases} \tilde{\mathbf{S}}_{\mathbf{12}} = \tilde{\mathbf{S}}_{\mathbf{21}} = \mathbf{0} \\ \tilde{\mathbf{S}}_{\mathbf{22}} = 1 - C \\ \tilde{\mathbf{S}}_{\mathbf{11}} = C \cdot \mathbf{P} + (1 - C) \cdot \mathbf{I}_n \end{cases}$$

Combing the above equations with Eq.(49) produces

$$\tilde{\mathbf{S}}_{\mathbf{11}} = C\hat{\mathbf{Q}}\tilde{\mathbf{S}}_{\mathbf{11}}\hat{\mathbf{Q}}^T + \tfrac{(1-C)C}{(d_j+1)^2}\mathbf{e}_j\mathbf{e}_j{}^T + (1-C)\mathbf{I}_n \qquad (50)$$

Applying $\tilde{\mathbf{S}}_{\mathbf{11}} = \mathbf{S} + \Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ and $\mathbf{S} = C\mathbf{Q}\mathbf{S}\mathbf{Q}^T + (1-C)\mathbf{I}_n$ to Eq.(50) and rearranging the terms, we have

$$\Delta\tilde{\mathbf{S}}_{\mathbf{11}} = C\hat{\mathbf{Q}}\Delta\tilde{\mathbf{S}}_{\mathbf{11}}\hat{\mathbf{Q}}^T + \tfrac{C}{d_j+1}\left(2\alpha\mathbf{e}_j\mathbf{e}_j{}^T - \mathbf{e}_j\mathbf{y}^T - \mathbf{y}\mathbf{e}_j{}^T\right)$$

with α and $\mathbf{y}$ being defined by Eq.(45). □

Theorem 8 implies that, in the case (C2), after a new edge (i, j) is inserted, the new SimRank matrix $\tilde{\mathbf{S}}$ takes an elegant diagonal block structure: the upper-left block of $\tilde{\mathbf{S}}$ is perturbed by $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ which is the solution to the rank-two Sylvester equation (47); the lower-right block of $\tilde{\mathbf{S}}$ is a constant $(1 - C)$. This structure of $\tilde{\mathbf{S}}$ suggests that the inserted edge $(i, j)_{i\notin V, j\in V}$ only has a recursive impact on the SimRanks with pairs $(x, y) \in V \times V$, but with no impacts on pairs $(x, y) \in (V \times \{i\}) \cup (\{i\} \times V)$. Thus, our incremental way of computing the new $\tilde{\mathbf{S}}$ will focus on the efficiency of obtaining $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ from Eq.(47). Fortunately, we notice that $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ satisfies the rank-two Sylvester equation, whose algebraic structure is similar to that of $\Delta\mathbf{S}$ in Eqs.(12) and (13) (in Section 4). Hence, our previous techniques to compute $\Delta\mathbf{S}$ in Eq.(13) can be analogously applied to compute $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ in Eq.(47), thus eliminating costly matrix-matrix multiplications, as will be illustrated in Algorithm 4.

One disadvantage of Theorem 8 is that, in order to get the auxiliary vector $\mathbf{z}$ for evaluating $\tilde{\mathbf{S}}$, one has to memorize the *entire* old matrix $\mathbf{S}$ in Eq.(45). In fact, we can utilize the technique of rearranging the terms of the SimRank equation (1) to characterize $\mathbf{Q}\mathbf{S}[\mathbf{Q}]^T_{j,\star}$ in terms of only one vector $[\mathbf{S}]_{\star,j}$ so as to avoid memorizing the entire $\mathbf{S}$, as shown below.

Theorem 9 *The auxiliary matrix $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ in Theorem 8 can be represented as*

$$\begin{aligned} &\Delta\tilde{\mathbf{S}}_{\mathbf{11}} = \tfrac{C}{d_j+1}\left(\mathbf{M} + \mathbf{M}^T\right) \ \textit{with} \\ &\mathbf{M} = \sum\nolimits_{k=0}^{\infty} C^k\hat{\mathbf{Q}}^k\mathbf{e}_j\mathbf{z}^T\left(\hat{\mathbf{Q}}^T\right)^k \end{aligned} \qquad (51)$$

where $\hat{\mathbf{Q}}$ is defined by Theorem 7 and

$$\mathbf{z} := \left(\tfrac{1}{2C(d_j+1)}\left([\mathbf{S}]_{j,j} - (1-C)^2\right) + \tfrac{1-C}{C}\right)\mathbf{e}_j - \tfrac{1}{C}[\mathbf{S}]_{\star,j} \qquad (52)$$

and $\mathbf{S}$ is the old SimRank matrix of G.

Algorithm 4: Inc-uSR-C2 $(G, (i, j), \mathbf{S}, K, C)$

Input : a directed graph $G = (V, E)$,
a new edge $(i, j)_{i\notin V,\ j\in V}$ inserted to G,
the old similarities $\mathbf{S}$ in G,
the number of iterations K,
the damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ in $G \cup \{(i, j)\}$.

initialize the transition matrix $\mathbf{Q}$ in G ;
$d_j :=$ in-degree of node j in G ;
$\mathbf{z} := \left(\frac{1}{2C(d_j+1)}\left([\mathbf{S}]_{j,j} - (1-C)^2\right) + \frac{1-C}{C}\right)\mathbf{e}_j - \frac{1}{C}[\mathbf{S}]_{\star,j}$;
initialize $\boldsymbol{\xi}_0 := \mathbf{e}_j, \quad \boldsymbol{\eta}_0 := \mathbf{z}, \quad \mathbf{M}_0 := \mathbf{e}_j\mathbf{z}^T$;
for $k = 0, 1, \cdots, K-1$ **do**
  $\boldsymbol{\xi}_{k+1} := C \cdot \mathbf{Q} \cdot \boldsymbol{\xi}_k - \frac{C}{d_j+1}([\mathbf{Q}]_{j,\star} \cdot \boldsymbol{\xi}_k) \cdot \mathbf{e}_j$;
  $\boldsymbol{\eta}_{k+1} := \mathbf{Q} \cdot \boldsymbol{\eta}_k - \frac{1}{d_j+1}([\mathbf{Q}]_{j,\star} \cdot \boldsymbol{\eta}_k) \cdot \mathbf{e}_j$;
  $\mathbf{M}_{k+1} := \boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T + \mathbf{M}_k$;
compute $\Delta\tilde{\mathbf{S}}_{\mathbf{11}} := \frac{C}{d_j+1}\left(\mathbf{M}_K + \mathbf{M}_K^T\right)$;
return $\tilde{\mathbf{S}} := \left[\begin{array}{c|c} \mathbf{S} + \Delta\tilde{\mathbf{S}}_{\mathbf{11}} & \mathbf{0} \\ \hline \mathbf{0} & 1-C \end{array}\right]$;

Proof We multiply the SimRank equation by $\mathbf{e}_j$ to get

$$[\mathbf{S}]_{\star,j} = C \cdot \mathbf{Q}\mathbf{S}[\mathbf{Q}]^T_{j,\star} + (1-C) \cdot \mathbf{e}_j.$$

Combining this with $\mathbf{y} = \mathbf{Q}\mathbf{S}[\mathbf{Q}]^T_{j,\star}$ in Eq.(45) produces

$$\mathbf{y} = \tfrac{1}{C}[\mathbf{S}]_{\star,j} - \tfrac{1-C}{C}\mathbf{e}_j \ \text{ and } \ \mathbf{y}_j = \tfrac{1}{C}[\mathbf{S}]_{j,j} - \tfrac{1-C}{C}.$$

Plugging these results into Eq.(45), we can get Eq.(52).

Also, the recursive form of $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ in Eq.(47) can be converted into the following series:

$$\begin{aligned} \Delta\tilde{\mathbf{S}}_{\mathbf{11}} &= \tfrac{C}{d_j+1}\sum\nolimits_{k=0}^{\infty} C^k\hat{\mathbf{Q}}^k\left(\mathbf{e}_j\mathbf{z}^T + \mathbf{z}\mathbf{e}_j{}^T\right)\left(\hat{\mathbf{Q}}^T\right)^k \\ &= \mathbf{M} + \mathbf{M}^T \end{aligned}$$

with $\mathbf{M}$ being defined by Eq.(51). □

For edge insertion of the case (C2), Theorem 9 gives an efficient method to compute the update matrix $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$. We note that the form of $\Delta\tilde{\mathbf{S}}_{\mathbf{11}}$ in Eq.(51) is similar to that of $\Delta\tilde{\mathbf{S}}$ in Eq.(25). Thus, similar to Theorem 3, the follow trick can be applied to compute the series of $\mathbf{M}$:

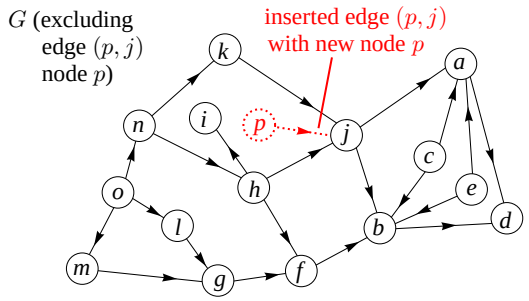
initialize $\boldsymbol{\xi}_0 \leftarrow \mathbf{e}_j, \quad \boldsymbol{\eta}_0 \leftarrow \mathbf{z}, \quad \mathbf{M}_0 \leftarrow \mathbf{e}_j \cdot \mathbf{z}^T$
for $k = 0, 1, 2, \cdots$
  $\boldsymbol{\xi}_{k+1} \leftarrow C \cdot \hat{\mathbf{Q}} \cdot \boldsymbol{\xi}_k, \quad \boldsymbol{\eta}_{k+1} \leftarrow \hat{\mathbf{Q}} \cdot \boldsymbol{\eta}_k$
  $\mathbf{M}_{k+1} \leftarrow \boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T + \mathbf{M}_k$

so as to avoid matrix-matrix multiplications.

Note that, in the above formulas, to avoid memorizing the auxiliary $\hat{\mathbf{Q}}$, we can compute $\hat{\mathbf{Q}} \cdot \boldsymbol{\xi}_k$ as follows:

$$\hat{\mathbf{Q}} \cdot \boldsymbol{\xi}_k = \mathbf{Q} \cdot \boldsymbol{\xi}_k - \tfrac{1}{d_j+1}([\mathbf{Q}]_{j,\star} \cdot \boldsymbol{\xi}_k) \cdot \mathbf{e}_j.$$

In Algorithm 4, we present the edge insertion of our method for the case (C2) to incrementally update new



Node-Pair	in G $\mathsf{sim}_{\mathsf{old}}$	in $G \cup \Delta G$ $\mathsf{sim}_{\mathsf{new}}$
(a,b)	0.0745	0.0596
$\boldsymbol{(f,i)}$	**0.2464**	**0.2464**
(f,j)	0.2064	0.1376
$\boldsymbol{(g,h)}$	**0.128**	**0.128**
$\boldsymbol{(g,k)}$	**0.128**	**0.128**
$\boldsymbol{(h,k)}$	**0.288**	**0.288**
(i,j)	0.3104	0.2069
$\boldsymbol{(l,m)}$	**0.16**	**0.16**
$\boldsymbol{(l,n)}$	**0.16**	**0.16**
$\boldsymbol{(m,n)}$	**0.16**	**0.16**

Fig. 4: Incrementally update SimRank when a new edge (p,j) with $p \notin V$ and $j \in V$ is inserted into $G=(V,E)$

SimRank scores. The total complexity of Algorithm 4 is $O(Kn^2)$ time and $O(n^2)$ memory in the worst case for retrieving all n^2 pairs of scores, which is dominated by Line 8. To reduce its computational time further, the similar pruning techniques in Section 5 can be applied to Algorithm 4. This can speed up the computational time to $O(K(nd+|\mathsf{AFF}|))$, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ is the size of "affected areas" in $\Delta\mathbf{S}_{11}$. In the next section, we shall also substantially reduce its memory from $O(n^2)$ to $O(nd)$.

Example 8 Consider the citation digraph G in Fig.4. If the new edge (j,p) with new node p is inserted to G, the new $\tilde{\mathbf{S}}$ can be incrementally derived from the old $\mathbf{S}$ as follows:

First, we obtain $\Delta\tilde{\mathbf{S}}_{11}$ according to Theorem 9. Note that $C=0.8$, $d_j=2$, and the old SimRank scores

$$[\mathbf{S}]_{\star,j} = [\overset{(a)}{0},\ \cdots,\ \overset{(e)}{0},\ \overset{(f)}{0.2064},\ \overset{(g)}{0},\ \overset{(h)}{0},\ \overset{(i)}{0.3104},\ \overset{(j)}{0.5104},\ \overset{(k)}{0},\ \cdots,\ \overset{(o)}{0}\ ]^T$$

It follows from Eq.(52) that the auxiliary vector

$$\mathbf{z} = \left(\tfrac{1}{2\times 0.8(2+1)}\left(0.5104-(1-0.8)^2\right)+\tfrac{1-0.8}{0.8}\right)\mathbf{e}_j - \tfrac{1}{0.8}[\mathbf{S}]_{\star,j}$$
$$= [\overset{(a)}{0},\ \cdots,\ \overset{(e)}{0},\ \overset{(f)}{-0.258},\ \overset{(g)}{0},\ \overset{(h)}{0},\ \overset{(i)}{-0.388},\ \overset{(j)}{-0.29},\ \overset{(k)}{0},\ \cdots,\ \overset{(o)}{0}\ ]^T$$

Utilizing $\mathbf{z}$, we can obtain $\mathbf{M}$ from Eq.(51). Thus, $\Delta\tilde{\mathbf{S}}_{11}$ can be computed from $\mathbf{M}$ as

$$\Delta\tilde{\mathbf{S}}_{11} = \tfrac{0.8}{2+1}\left(\mathbf{M}+\mathbf{M}^T\right) =$$

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)⋯(o)
(a)	−0.0137	−0.0149	0	0							
(b)	−0.0149	−0.0146	0	0			0			0	0
(c)	0	0	0	0							
(d)	0	0	0	−0.0116							
(e)										0	
(f)										−0.0688	
(g)		0					0			0	0
(h)										0	
(i)										−0.1035	
(j)		0			0	−0.0688	0	0	−0.1035	−0.1547	0
⋮ (o)		0					0			0	0

Next, by Theorem 8, we obtain the new SimRank

$$\tilde{\mathbf{S}} = \left[\begin{array}{c|c} \mathbf{S}+\Delta\tilde{\mathbf{S}}_{11} & \mathbf{0} \\ \hline \mathbf{0} & 0.2 \end{array}\right]$$

which is partially illustrated in Fig.4. □

6.3 Inserting an edge (i,j) with $i \notin V$ and $j \notin V$

We next focus on the case (C3), the insertion of an edge (i,j) with $i \notin V$ and $j \notin V$. Without loss of generality, it can be tacitly assumed that nodes i and j are indexed by $n+1$ and $n+2$, respectively. In this case, the inserted edge (i,j) accompanies the insertion of two new nodes, which can form another independent component in the new graph.

In this case, the new transition matrix $\tilde{\mathbf{Q}}$ can be characterized as a block diagonal matrix

$$\tilde{\mathbf{Q}} = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{N} \end{array}\right] \begin{array}{l} \} \ n \text{ rows} \\ \} \ 2 \text{ rows} \end{array} \quad \text{with} \quad \mathbf{N} := \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}.$$

With this structure, we can infer that the new SimRank matrix $\tilde{\mathbf{S}}$ takes the block diagonal form as

$$\tilde{\mathbf{S}} = \left[\begin{array}{c|c} \mathbf{S} & \mathbf{0} \\ \hline \mathbf{0} & \hat{\mathbf{S}} \end{array}\right] \begin{array}{l} \} \ n \text{ rows} \\ \} \ 2 \text{ rows} \end{array} \quad \text{with} \quad \hat{\mathbf{S}} \in \mathbb{R}^{2\times 2}.$$

This is because, after a new edge $(i,j)_{i\notin V, j\notin V}$ is added, all node-pairs $(x,y) \in (V\times\{i,j\} \cup \{i,j\}\times V)$ have zero SimRank scores since there are no connections between nodes x and y. Besides, the inserted edge (i,j) is an independent component that has no impact on $s(x,y)$ for $\forall (x,y) \in V\times V$. Hence, the submatrix $\hat{\mathbf{S}}$ of the new SimRank matrix can be derived by solving the equation:

$$\hat{\mathbf{S}} = C\cdot\mathbf{N}\cdot\hat{\mathbf{S}}\cdot\mathbf{N}^T + (1-C)\cdot\mathbf{I}_2 \quad \Rightarrow \quad \hat{\mathbf{S}} = \begin{bmatrix} 1-C & 0 \\ 0 & 1-C^2 \end{bmatrix}$$

This suggests that, for unit insertion of the case (C3), the new SimRank matrix becomes

$$\tilde{\mathbf{S}} = \left[\begin{array}{c|c} \mathbf{S} & \mathbf{0} \\ \hline \mathbf{0} & \hat{\mathbf{S}} \end{array}\right] \in \mathbb{R}^{(n+2)\times(n+2)} \quad \text{with} \quad \hat{\mathbf{S}} = \begin{bmatrix} 1-C & 0 \\ 0 & 1-C^2 \end{bmatrix}.$$

Algorithm 5 presents our incremental method to obtain the new SimRank matrix $\tilde{\mathbf{S}}$ for edge insertion of the case (C3), which requires just $O(1)$ time.

Algorithm 5: Inc-uSR-C3 $(G,(i,j),\mathbf{S},C)$

Input : a directed graph $G=(V,E)$,
a new edge $(i,j)_{i\notin V,\ j\notin V}$ inserted to G,
the old similarities $\mathbf{S}$ in G,
the damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$.

1 compute $\hat{\mathbf{S}} := \begin{bmatrix} 1-C & 0 \\ 0 & 1-C^2 \end{bmatrix}$;

2 **return** $\tilde{\mathbf{S}} := \left[\begin{array}{c|c} \mathbf{S} & \mathbf{0} \\ \hline \mathbf{0} & \hat{\mathbf{S}} \end{array}\right]$;

7 Memory Efficiency

In previous sections, our main focus is devoted to speeding up the computational time of incremental SimRank. However, for evaluating all pairs of SimRank scores, the memory requirement for Algorithms 1–5 is still $O(n^2)$ in the worst case because all pairs scores are computed and stored at the same time. In this section, we propose a novel partitioning technique that can update all pairs of SimRanks column by column in just $O(dn)$ memory, without any loss of accuracy and extra running time.

Let us first analyze the $O(n^2)$ memory requirement for Algorithms 1–5 in Sections 4–6. We notice that there are two factors dominating the original $O(n^2)$ memory: (1) the storage of the entire $n\times n$ old SimRank matrix $\mathbf{S}$, and (2) the computation of $\mathbf{M}_k$ from one outer product (in Line 14 of Algorithm 1, and Line 8 of Algorithm 4). Apart from the storage of the old $\mathbf{S}$ and $\mathbf{M}_k$, the space required for the rest of steps is dominated by $O(dn)$, including (a) the storage of the sparse matrix $\mathbf{Q}$ and (b) the sparse matrix-vector multiplications.

To overcome the bottleneck of the $O(n^2)$ memory, our central idea is to update all pairs of new SimRank scores in a column-by-column fashion, without the need to load the entire matrices of $\mathbf{S}$ and $\mathbf{M}_k$ into memory. Precisely, we can first partition the new entire SimRank matrix $\tilde{\mathbf{S}}$ into column vectors $[\tilde{\mathbf{S}}]_{\star,1}, [\tilde{\mathbf{S}}]_{\star,2}, \cdots$, and then update every column $[\tilde{\mathbf{S}}]_{\star,x}$ separately.

For example, for insertion case (C1) in Subsection 6.1, it is not necessary to compute all columns of new $\tilde{\mathbf{S}}$ simultaneously by storing the *entire* old $\mathbf{S}$ in Algorithm 3. Instead, we can compute $\tilde{\mathbf{S}}$ (Line 4) column by column:

$$\tilde{\mathbf{S}} := \left[\begin{array}{c|c} \overbrace{\mathbf{S}}^{n \text{ cols}} & \overbrace{\mathbf{y}}^{\text{col } j} \\ \hline \mathbf{y}^T & z \end{array}\right] \quad \Rightarrow$$

memoize $\mathbf{y}$ and z
for each $x \leftarrow 1,\cdots,n$
  set $[\tilde{\mathbf{S}}]_{\star,x} \leftarrow \left[\begin{array}{c} [\mathbf{S}]_{\star,x} \\ \hline [\mathbf{y}]_x \end{array}\right]$
set $[\tilde{\mathbf{S}}]_{\star,n+1} \leftarrow \left[\begin{array}{c} \mathbf{y} \\ \hline z \end{array}\right]$

Similarly, one can modify Lines 15, 10, 2, respectively, in Algorithms 1, 4, 5 into a column-by-column style.

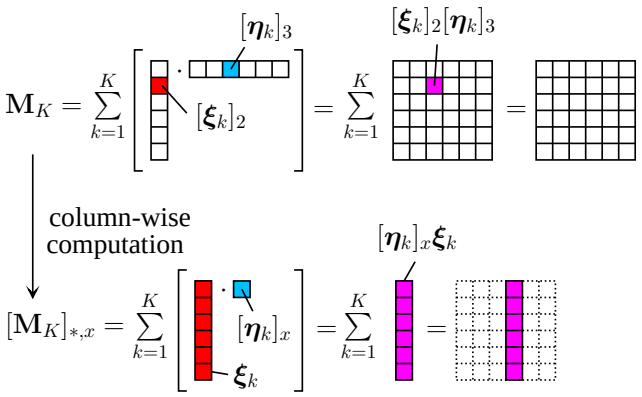


Fig. 5: Memory usage reduction by partitioning $\mathbf{M}_K$ in a column-by-column style

In this way, although Algorithms 3 and 5 can achieve $O(dn)$ memory, Algorithms 1 and 4 still require $O(n^2)$. The reason is that the computation of $\mathbf{M}_k$ in Line 14 (*resp.* 8) dominates the memory of Algorithm 1 (*resp.* 4). To resolve this problem, we also need split the computation of the entire $\mathbf{M}_k$ in a column-by-column fashion. Note that $\mathbf{M}_k$ in Line 14, Algorithm 1 and $\mathbf{M}_k$ in Line 8, Algorithm 4 exhibit a similar structure, *i.e.,* $\mathbf{M}_k$ is the summation of the outer product of two vectors:

for $k \leftarrow 0,\cdots,K-1$
  set $\mathbf{M}_{k+1} \leftarrow \boldsymbol{\xi}_{k+1}\cdot\boldsymbol{\eta}_{k+1}^T + \mathbf{M}_k$
update $\tilde{\mathbf{S}} \leftarrow \mathbf{S} + \mathbf{M}_K + \mathbf{M}_K^T$

Thus, we can split the above computation column-wisely:

for $x \leftarrow 1,\cdots,n$
  initialize $[\tilde{\mathbf{S}}]_{\star,x} \leftarrow [\mathbf{S}]_{\star,x}$, $\mathbf{y} \leftarrow \mathbf{0}$, $\mathbf{z} \leftarrow \mathbf{0}$
  for $k \leftarrow 0,\cdots,K-1$
    set $\mathbf{y} \leftarrow [\boldsymbol{\eta}_{k+1}]_x \cdot \boldsymbol{\xi}_{k+1} + \mathbf{y}$
    set $\mathbf{z} \leftarrow [\boldsymbol{\xi}_{k+1}]_x \cdot \boldsymbol{\eta}_{k+1} + \mathbf{z}$
    update $[\tilde{\mathbf{S}}]_{\star,x} \leftarrow [\tilde{\mathbf{S}}]_{\star,x} + \mathbf{y} + \mathbf{z}$

The main advantage of our revision is that, throughout the entire updating process, we need not store the entire matrix $\mathbf{M}_k$ and $\mathbf{S}$, and thereby, significantly reduce the memory usage from $O(n^2)$ to $O(dn)$.

Correctness. By applying successive substitution to the **for**-loop, we can verify that, in our above methods, every new score $[\tilde{\mathbf{S}}]_{u,v}$ has the consistent result[20]:

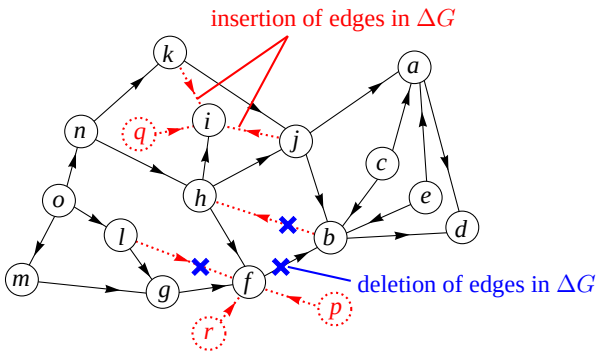
$$[\tilde{\mathbf{S}}]_{u,v} = [\mathbf{S}]_{u,v} + \sum_{k=1}^{K} [\boldsymbol{\xi}_k]_u \cdot [\boldsymbol{\eta}_k]_v + \sum_{k=1}^{K} [\boldsymbol{\xi}_k]_v \cdot [\boldsymbol{\eta}_k]_u$$

Thus, our partitioning approach does not compromise accuracy for high memory efficiency, which is achieved by the separable structure of $\mathbf{M}_K$ described as the sum of rank-one matrices, as pictorially depicted in Fig. 5.

[20] Here, our main focus is devoted to $\tilde{\mathbf{S}}$ in Algorithm 1. Note that similar techniques can be applied to $\tilde{\mathbf{S}}_{\mathbf{11}}$ in Algorithm 4.

G (solid line)

$\Delta G = \{(q,i,+),\ (b,h,+),\ (f,b,-),\ (l,f,+),\ (p,f,+),\ (l,f,-),\ (j,i,+),\ (r,f,+),\ (b,h,-),\ (k,i,+)\}$



Node-Pair	in G $\mathsf{sim}_{\mathsf{old}}$	in $G \oplus \Delta G$ $\mathsf{sim}_{\mathsf{new}}$
(a,b)	0.0745	0.0809
(a,i)	0	0.0340
(b,i)	0	0.0340
(f,i)	0.2464	0.0516
(f,j)	0.2064	0.1032
$\boldsymbol{(g,h)}$	**0.128**	**0.128**
$\boldsymbol{(g,k)}$	**0.128**	**0.128**
(h,k)	0.2880	0.2880
(i,j)	0.3104	0.1552
$\boldsymbol{(l,m)}$	**0.16**	**0.16**
$\boldsymbol{(l,n)}$	**0.16**	**0.16**
$\boldsymbol{(m,n)}$	**0.16**	**0.16**

Fig. 6: Batch updates for incremental SimRank when a sequence of edges ΔG are updated to $G = (V, E)$

8 Batch Updates

In this section, we consider the batch updates problem for incremental SimRank, *i.e.,* given an old graph $G = (V, E)$ and a sequence of edges ΔG to be updated to G, the retrieval of new SimRank scores in $G \oplus \Delta G$. Here, the set ΔG can be mixed with insertions and deletions:

$$\Delta G := \{(i_1, j_1, \mathsf{op}_1), (i_2, j_2, \mathsf{op}_2), \cdots, (i_{|\Delta G|}, j_{|\Delta G|}, \mathsf{op}_{|\Delta G|})\}$$

where (i_q, j_q) is the q-th edge in ΔG to be inserted into (if $\mathsf{op}_q =$"+") or deleted from (if $\mathsf{op}_q =$"−") G.

The straightforward approach to this problem is to update each edge of ΔG one by one, by running a unit update algorithm for $|\Delta G|$ times. However, this would produce many unnecessary intermediate results and redundant updates that may cancel out each other.

Example 9 Consider the old citation graph G in Fig. 6, and a sequence of edge updates ΔG to G:

$$\Delta G = \{(q,i,+),\ \boldsymbol{(b,h,+)},\ (f,b,-),\ \boldsymbol{(l,f,+)},\ (p,f,+),\ \boldsymbol{(l,f,-)},\ (j,i,+),\ (r,f,+),\ \boldsymbol{(b,h,-)},\ (k,i,+)\}$$

We notice that, in ΔG, the edge insertion $(b,h,+)$ can cancel out the edge deletion $(b,h,-)$. Similarly, $(l,f,+)$ can cancel out $(l,f,-)$. Thus, after edges cancellation, the *net* update of ΔG, denoted as ΔG_{net}, is

$$\Delta G_{\text{net}} = \{(q,i,+),\ (f,b,-),\ (p,f,+),\ (j,i,+),\ (r,f,+),\ (k,i,+)\} \qquad \square$$

Example 9 suggests that a portion of redundancy in ΔG arises from the insertion and deletion of the same edge that may cancel out each other. After cancellation, it is easy to verify that

$$|\Delta G_{\text{net}}| \leq |\Delta G| \quad \text{yet} \quad G \oplus \Delta G_{\text{net}} = G \oplus \Delta G.$$

To obtain ΔG_{net} from ΔG, we can readily use hashing techniques to count occurrences of updates in ΔG. More specifically, we use each edge of ΔG as a hash key, and initialize each key with zero count. Then, we scan each edge of ΔG once, and increment (*resp.* decrement) its count by one each time an edge insertion (*resp.* deletion) appears in ΔG. After all edges in ΔG are scanned, the edges whose counts are nonzeros make a net update ΔG_{net}. All edges in ΔG_{net} with $+1$ (*resp.* -1) counts make a net insertion update $\Delta G^{+}_{\text{net}}$ (*resp.* a net deletion update $\Delta G^{-}_{\text{net}}$). Clearly, we have

$$\Delta G_{\text{net}} = \Delta G^{+}_{\text{net}} \cup \Delta G^{-}_{\text{net}}.$$

Having reduced ΔG to the net edge updates ΔG_{net}, we next merge the updates of "similar sink edges" in ΔG_{net} to speedup the batch updates further.

We first introduce the notion of "similar sink edges".

Definition 2 Two distinct edges (a,c) and (b,c) are called "similar sink edges" *w.r.t.* node c if they have a common end node c that both a and b point to.

"Similar sink edges" is introduced to partition ΔG_{net}. To be specific, we first sort all the edges $\{(i_p, j_p)\}$ of $\Delta G^{+}_{\text{net}}$ (*resp.* $\Delta G^{-}_{\text{net}}$) according to its end node j_p. Then, the "similar sink edges" *w.r.t.* node j_p form a partition of $\Delta G^{+}_{\text{net}}$ (*resp.* $\Delta G^{-}_{\text{net}}$). For each block $\{(i_{p_k}, j_p)\}$ in $\Delta G^{+}_{\text{net}}$, we next split it further into two sub-blocks according to whether its end node i_{p_k} is in the old V. Thus, after partitioning, each block in $\Delta G^{+}_{\text{net}}$ (*resp.* $\Delta G^{-}_{\text{net}}$), denoted as

$$\{(i_1, j),\ (i_2, j),\ \cdots,\ (i_\delta, j)\},$$

falls into one of the following cases:

(C0) $i_1 \in V,\ i_2 \in V,\ \cdots, i_\delta \in V$ and $j \in V$;
(C1) $i_1 \in V,\ i_2 \in V,\ \cdots, i_\delta \in V$ and $j \notin V$;
(C2) $i_1 \notin V,\ i_2 \notin V,\ \cdots, i_\delta \notin V$ and $j \in V$;
(C3) $i_1 \notin V,\ i_2 \notin V,\ \cdots, i_\delta \notin V$ and $j \notin V$.

Example 10 Let us recall ΔG_{net} derived by Example 9, in which $\Delta G_{\text{net}} = \Delta G^{+}_{\text{net}} \cup \Delta G^{-}_{\text{net}}$ with

$$\Delta G^{+}_{\text{net}} = \{(q,i,+),\ (p,f,+),\ (j,i,+),\ (r,f,+),\ (k,i,+)\}$$
$$\Delta G^{-}_{\text{net}} = \{(f,b,-)\}.$$

	when	new transition matrix $\tilde{\mathbf{Q}}$	new SimRank matrix $\tilde{\mathbf{S}}$
without new nodes insertion	(C0) insert $i_1 \in V$ $\cdots$ $i_\delta \in V$ $j \in V$	$\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T$ with $\mathbf{u} := \begin{cases} \mathbf{e}_j & (d_j = 0) \\ \frac{\delta}{d_j+\delta}\mathbf{e}_j & (d_j > 0) \end{cases}$, $\mathbf{v} := \begin{cases} \frac{1}{\delta}\mathbf{e}_I & (d_j = 0) \\ \frac{1}{\delta}\mathbf{e}_I - [\mathbf{Q}]^T_{j,\star} & (d_j > 0) \end{cases}$	$\mathbf{\Delta S} = \mathbf{M} + \mathbf{M}^T$ with $\mathbf{M} := \sum_{k=0}^{\infty} C^{k+1}\tilde{\mathbf{Q}}^k \mathbf{e}_j \boldsymbol{\gamma}^T (\tilde{\mathbf{Q}}^T)^k$, $\boldsymbol{\gamma} := \begin{cases} \frac{1}{\delta}\mathbf{Q}\cdot[\mathbf{S}]_{\star,I} + \frac{1}{2\delta^2}[\mathbf{S}]_{I,I}\cdot\mathbf{e}_j & (d_j = 0) \\ \frac{\delta}{(d_j+\delta)}\left(\frac{1}{\delta}\mathbf{Q}\cdot[\mathbf{S}]_{\star,I} - \frac{1}{C}\cdot[\mathbf{S}]_{\star,j} + (\frac{\lambda\delta}{2(d_j+\delta)} + \frac{1}{C} - 1)\cdot\mathbf{e}_j\right) & (d_j > 0) \end{cases}$, $\lambda := \frac{1}{\delta^2}[\mathbf{S}]_{I,I} + \frac{1}{C}\cdot[\mathbf{S}]_{j,j} - \frac{2}{\delta}\cdot[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,I} - \frac{1}{C} + 1$
	(C0) delete $i_1 \in V$ $\cdots$ $i_\delta \in V$ $j \in V$	$\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T$ with $\mathbf{u} := \begin{cases} \mathbf{e}_j & (d_j = 1) \\ \frac{\delta}{d_j-\delta}\mathbf{e}_j & (d_j > 1) \end{cases}$, $\mathbf{v} := \begin{cases} -\frac{1}{\delta}\mathbf{e}_I & (d_j = 1) \\ [\mathbf{Q}]^T_{j,\star} - \frac{1}{\delta}\mathbf{e}_I & (d_j > 1) \end{cases}$	$\mathbf{\Delta S} = \mathbf{M} + \mathbf{M}^T$ with $\mathbf{M} := \sum_{k=0}^{\infty} C^{k+1}\tilde{\mathbf{Q}}^k \mathbf{e}_j \boldsymbol{\gamma}^T (\tilde{\mathbf{Q}}^T)^k$, $\boldsymbol{\gamma} := \begin{cases} -\frac{1}{\delta}\mathbf{Q}\cdot[\mathbf{S}]_{\star,I} + \frac{1}{2\delta^2}[\mathbf{S}]_{I,I}\cdot\mathbf{e}_j & (d_j = 1) \\ \frac{\delta}{(d_j-\delta)}\left(\frac{1}{C}\cdot[\mathbf{S}]_{\star,j} - \frac{1}{\delta}\mathbf{Q}\cdot[\mathbf{S}]_{\star,I} + (\frac{\lambda\delta}{2(d_j-\delta)} - \frac{1}{C} + 1)\cdot\mathbf{e}_j\right) & (d_j > 1) \end{cases}$, $\lambda := \frac{1}{\delta^2}[\mathbf{S}]_{I,I} + \frac{1}{C}\cdot[\mathbf{S}]_{j,j} - \frac{2}{\delta}\cdot[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,I} - \frac{1}{C} + 1$
with new nodes insertion	(C1) insert $i_1 \in V$ $\cdots$ $i_\delta \in V$ $j \notin V$	$\tilde{\mathbf{Q}} = \left[\begin{array}{c\|c} \mathbf{Q} & \mathbf{0} \\ \hline \frac{1}{\delta}\mathbf{e}_I^T & 0 \end{array}\right]$ $\}$ n rows $\to$ row j	$\tilde{\mathbf{S}} = \left[\begin{array}{c\|c} \mathbf{S} & \mathbf{y} \\ \hline \mathbf{y}^T & \frac{C}{\delta^2}[\mathbf{S}]_{I,I} + (1-C) \end{array}\right]$ $\}$ n rows $\to$ row j with $\mathbf{y} := \frac{C}{\delta}\mathbf{Q}[\mathbf{S}]_{\star,I}$
	(C2) insert $i_1 \notin V$ $\cdots$ $i_\delta \notin V$ $j \in V$	$\tilde{\mathbf{Q}} = \left[\begin{array}{c\|c} \hat{\mathbf{Q}} & \frac{1}{d_j+\delta}\mathbf{e}_j\mathbf{1}_\delta^T \\ \hline \mathbf{0} & 0 \end{array}\right]$ $\}$ n rows $\}$ δ rows with $\hat{\mathbf{Q}} := \mathbf{Q} - \frac{\delta}{d_j+\delta}\mathbf{e}_j[\mathbf{Q}]_{j,\star}$	$\tilde{\mathbf{S}} = \left[\begin{array}{c\|c} \mathbf{S} + \frac{C\delta}{d_j+\delta}\left(\mathbf{M}+\mathbf{M}^T\right) & \mathbf{0} \\ \hline \mathbf{0} & (1-C)\mathbf{I}_\delta \end{array}\right]$ $\}$ n rows $\}$ δ rows with $\mathbf{M} := \sum_{k=0}^{\infty} C^k\hat{\mathbf{Q}}^k\mathbf{e}_j\mathbf{z}^T(\hat{\mathbf{Q}}^T)^k$, $\mathbf{z} := \left(\frac{1}{2C(d_j+\delta)}\left(\delta[\mathbf{S}]_{j,j} - (\delta-C)(1-C)\right) + \frac{1-C}{C}\right)\mathbf{e}_j - \frac{1}{C}[\mathbf{S}]_{\star,j}$
	(C3) insert $i_1 \notin V$ $\cdots$ $i_\delta \notin V$ $j \notin V$	$\tilde{\mathbf{Q}} = \left[\begin{array}{c\|c} \mathbf{Q} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{N} \end{array}\right]$ $\}$ n rows $\}$ $\delta+1$ rows with $\mathbf{N} := \left[\begin{array}{c\|c} \mathbf{0} & \mathbf{0} \\ \hline \frac{1}{\delta}\mathbf{1}_\delta^T & 0 \end{array}\right]$ $\}$ δ rows $\to$ row j	$\tilde{\mathbf{S}} = \left[\begin{array}{c\|c} \mathbf{S} & \mathbf{0} \\ \hline \mathbf{0} & \hat{\mathbf{S}} \end{array}\right]$ $\}$ n rows $\}$ $\delta+1$ rows with $\hat{\mathbf{S}} := \left[\begin{array}{c\|c} (1-C)\mathbf{I}_\delta & \mathbf{0} \\ \hline \mathbf{0} & (1-C)(1+\frac{C}{\delta}) \end{array}\right]$ $\}$ δ rows $\to$ row j

Table 2: Batch updates for a sequence of edges $\{(i_1, j), \cdots, (i_\delta, j)\}$ to the old graph $G = (V, E)$, where $[\mathbf{S}]_{\star,I} := \sum_{i\in I}[\mathbf{S}]_{\star,i}$, $[\mathbf{S}]_{I,I} := \sum_{i\in I}[\mathbf{S}]_{i,I}$, $\mathbf{1}_\delta := (1, 1, \cdots, 1)^T \in \mathbb{R}^{\delta\times 1}$

We first partition ΔG^+_{net} by "similar sink edges" into

$$\Delta G^+_{\text{net}} = \{(q, i, +),\ (j, i, +),\ (k, i, +)\} \cup \{(p, f, +),\ (r, f, +)\}$$

In the first block of ΔG^+_{net}, since the nodes $q \notin V$, $j \in V$, and $k \in V$, we will partition this block further into $\{(q, i, +)\} \cup \{(j, i, +), (k, i, +)\}$. Eventually,

$$\Delta G^+_{\text{net}} = \{(q, i, +)\} \cup \{(j, i, +), (k, i, +)\} \cup \{(p, f, +), (r, f, +)\} \quad \square$$

The main advantage of our partitioning approach is that, after partition, all the edge updates in each block can be processed simultaneously, instead of one by one. To elaborate on this, we use case (C0) as an example, *i.e.,* the insertion of δ edges $\{(i_1, j),\ (i_2, j),\ \cdots,\ (i_\delta, j)\}$ into $G = (V, E)$ when $i_1 \in V, \cdots, i_\delta \in V$, and $j \in V$. Analogous to Theorem 1, one can readily prove that, after such δ edges are inserted, the changes $\mathbf{\Delta Q}$ to the old transition matrix is still a *rank-one* matrix that can be decomposed as

$$\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T \quad \text{with}$$

$$\mathbf{u} := \begin{cases} \mathbf{e}_j & (d_j = 0) \\ \frac{\delta}{d_j+\delta}\mathbf{e}_j & (d_j > 0) \end{cases}, \quad \mathbf{v} := \begin{cases} \frac{1}{\delta}\mathbf{e}_I & (d_j = 0) \\ \frac{1}{\delta}\mathbf{e}_I - [\mathbf{Q}]^T_{j,\star} & (d_j > 0) \end{cases}$$

where $\mathbf{e}_I$ is an $n\times 1$ vector with its entry $[\mathbf{e}_I]_x = 1$ if $x \in I \triangleq \{i_1, i_2, \cdots, i_\delta\}$, and $[\mathbf{e}_I]_x = 0$ if $x \notin V$. Since the rank-one structure of $\mathbf{\Delta Q}$ is preserved for updating δ edges, Theorem 2 still holds under the new settings of $\mathbf{u}$ and $\mathbf{v}$ for batch updates. Therefore, the changes $\mathbf{\Delta S}$ to the SimRank matrix in response to δ edges insertion can be represented as a similar formulation to Theorem 3, as illustrated in the first row of Table 2. Similarly, we

Algorithm 6: Inc-bSR $(G,(i,j),\mathbf{S},C)$

Input : a directed graph $G=(V,E)$,
a sequence of edge updates $\Delta G=\{(i,j,\mathsf{op})\}$,
the old similarities $\mathbf{S}$ in G,
the damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ in $G\oplus\Delta G$.

1 obtain the net update $\Delta G_{\rm net}$ from ΔG via hashing ;
2 split $\Delta G_{\rm net}=\Delta G^{+}_{\rm net}\cup\Delta G^{-}_{\rm net}$ according to op ;
3 partition $\Delta G^{+}_{\rm net}$ and $\Delta G^{-}_{\rm net}$ by "similar sink edges" ;
4 **for** *each block of* ΔG^{+}_{net} **do**
5 split all edges $\{(i,j)\}$ of each block further into (at most) two sub-blocks based on whether $i\in V$
6 **for** *each block of* ΔG^{-}_{net} **do**
7 delete all edges of each block and update $\tilde{\mathbf{S}}$ via Table 2 ;
8 remove all singleton nodes in the graph;
9 **for** *each sub-block of* ΔG^{+}_{net} **do**
10 insert all edges of each sub-block and update $\tilde{\mathbf{S}}$ via Table 2 ;
11 **return** $\tilde{\mathbf{S}}$;

can also extend Theorems 6–9 in Section 6 to support batch updates of δ edges for other cases (C1)–(C3) that accompany new node(s) insertion. Table 2 summarizes the new $\mathbf{Q}$ and $\mathbf{S}$ in response to such batch edge updates of all the cases. When $\delta=1$, these batch update results in Table 2 can be reduced to the unit update results of Theorems 1–9.

Algorithm 6 presents an efficient batch updates algorithm, Inc-bSR, for dynamical SimRank computation. The actual computational time of Inc-bSR depends on the input parameter ΔG since different update types in Table 2 would result in different computational time. However, we can readily show that Inc-bSR is superior to the $|\Delta G|$ executions of the unit update algorithm, because Inc-bSR can process the "similar sink updates" of each block simultaneously and can cancel out redundant updates. To clarify this, let us assume that $|\Delta G_{\rm net}|$ can be partitioned into $|B|$ blocks, with δ_t denoting the number of edge updates in t-th block. In the worst case, we assume that all edge updates happen to be the most time-consuming case (C0) or (C2). Then, the total time for handling $|\Delta G|$ updates is bounded by

$$\begin{aligned}&O\Big(\sum_{t=1}^{|B|}\big(n\delta_t+\delta_t^2+K(nd+\delta_t+|\mathsf{AFF}|)\big)\Big)\\ \le&O\Big(n|\Delta G_{\rm net}|+|\Delta G_{\rm net}|\sum_{t=1}^{|B|}\delta_t+K\sum_{t=1}^{|B|}(nd+\delta_t+|\mathsf{AFF}|)\Big)\\ \le&O\big((n+|\Delta G_{\rm net}|)|\Delta G_{\rm net}|+K(|B|nd+|\Delta G_{\rm net}|+|B||\mathsf{AFF}|)\big)\end{aligned}$$

Note that $|B|\le|\Delta G_{\rm net}|$, in general $|B|\ll|\Delta G_{\rm net}|$. Thus, Inc-bSR is typically much faster than the $|\Delta G|$ executions of the unit update algorithm that is bounded by $O\big(|\Delta G|K(nd+\Delta G+|\mathsf{AFF}|)\big)$.

The memory usage of Inc-bSR, if we incorporate the column-wise technique of Section 7, can be bounded by $O(n(\max_{t=1}^{|B|}\delta_t)+nd)$ in the worst case, because $O(n\delta_t)$ memory is needed to store δ_t columns of $\mathbf{S}$ when $[\mathbf{S}]_{\star,I}$ is required for the t-th block.

Example 11 Recall from Example 9 that a sequence of edge updates ΔG to the graph $G=(V,E)$ in Fig. 6. We want to compute new SimRank scores in $G\oplus\Delta G$.

First, we can use hashing method to obtain the net update $\Delta G_{\rm net}$ from ΔG, as shown in Example 9.

Next, by Example 10, we can partition $\Delta G_{\rm net}$ into

$$\Delta G^{+}_{\rm net}=\{(q,i,+)\}\cup\{(j,i,+),(k,i,+)\}\cup\{(p,f,+),(r,f,+)\}$$
$$\Delta G^{-}_{\rm net}=\{(f,b,-)\}$$

Then, for each block, we can apply the formulae in Table 2 to update all edges simultaneously in a batch fashion. The results are partially depicted as follows:

Node Pairs	$\mathsf{sim}_{\mathsf{old}}$ in G	$(f,b,-)$	$(q,i,+)$	$(j,i,+)$ $(k,i,+)$	$(p,f,+)$ $(r,f,+)$
(a,b)	0.0745	0.0809	0.0809	0.0809	0.0809
(a,i)	0	0	0	0.0340	0.0340
(b,i)	0	0	0	0.0340	0.0340
(f,i)	0.2464	0.2464	0.1232	0.1032	0.0516
(f,j)	0.2064	0.2064	0.2064	0.2064	0.1032
(g,h)	0.128	0.128	0.128	0.128	0.128
(g,k)	0.128	0.128	0.128	0.128	0.128
(h,k)	0.288	0.288	0.288	0.288	0.288
(i,j)	0.3104	0.3104	0.1552	0.1552	0.1552
(l,m)	0.16	0.16	0.16	0.16	0.16
(l,n)	0.16	0.16	0.16	0.16	0.16
(m,n)	0.16	0.16	0.16	0.16	0.16

The column '$(q,i,+)$' represents the updated SimRank scores after the edge (q,i) is added to $G\oplus\{(f,b,-)\}$. The last column is the new SimRanks in $G\oplus\Delta G$. □

9 Experimental Evaluation

In this section, we present a comprehensive experimental study on real and synthetic datasets, to demonstrate (i) the fast computational time of Inc-SR to incrementally update SimRanks on large time-varying networks, (ii) the pruning power of Inc-SR that can discard unnecessary incremental updates outside "affected areas"; (iii) the exactness of Inc-SR, as compared with Inc-SVD; (iv) the high efficiency of our complete scheme that integrates Inc-SR with Inc-uSR-C1, Inc-uSR-C2, Inc-uSR-C3 to support link updates that allow new nodes insertion; (v) the small memory usage of our improved version in Section 7 to incrementally evaluate all pairs of scores; (vi) the fast computation time of our batch update algorithm Inc-bSR against the unit update method Inc-SR.

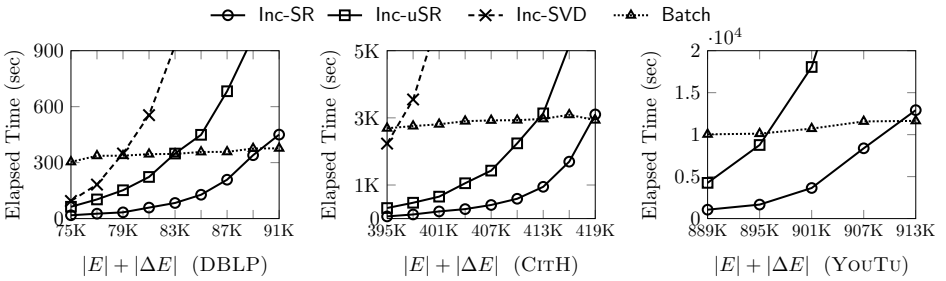


Fig. 7: Time Efficiency on Real Data (ΔE does not accompany new nodes)

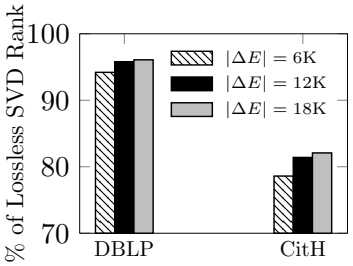


Fig. 8: % of Lossless SVD Rank

9.1 Experimental Settings

Datasets. We adopt both real and synthetic datasets. The real datasets include DBLP,CitH, and YouTu.

(1) DBLP[21] is a co-citation graph, where each node is a paper with attributes (*e.g.,* publication year), and edges are citations. By virtue of the publication year, we extract several snapshots, each consisting of 93,560 edges and 13,634 nodes.

(2) CitH[22] is a reference network (cit-HepPh) from e-Arxiv. If a paper u references v, there is one link from u to v. The dataset has 421,578 edges and 34,546 nodes.

(3) YouTu[23] is a YouTube network, where a video u (node) is linked to v if v is in the relevant video list of u. We extract snapshots according to the age of the videos, and each has 953,534 edges and 178,470 nodes.

To generate synthetic graphs and updates, we adopt GraphGen[24] generation engine. The graphs are controlled by (a) the number of nodes $|V|$, and (b) the number of edges $|E|$. We produce a sequence of graphs that follow the linkage generation model [7]. To control graph updates, we use two parameters simulating real evolution: (a) update type (edge/node insertion or deletion), and (b) the size of updates $|\Delta G|$.

Algorithms. We implement the following algorithms: (a) Inc-SVD, the SVD-based link-update algorithm [12]; (b) Inc-uSR, our incremental method without pruning; (c) Batch, the batch SimRank method via fine-grained memoization [20]; (d) Inc-SR, our incremental method with pruning power but not supporting nodes insertion; (e) Inc-SR-All, our complete enhanced version of Inc-SR that allows nodes insertion by incorporating Inc-uSR-C1, Inc-uSR-C2, and Inc-uSR-C3; (f) Inc-bSR, our batch incremental update version of Inc-SR.

Parameters. We set the decay factor $C = 0.6$, as used by the prior work [9]. The default iteration number is set to $K = 15$, with which a high accuracy $C^K \leq 0.0005$ is guaranteed [14]; on big CitH and YouTu, we choose $K = 10$, as previously used by [9]. The target rank r for Inc-SVD is a speed-accuracy trade-off — as shown in the experiments of [12], the highest speedup is achieved when $r = 5$. Thus, in the time evaluation, $r = 5$ is used. In the exactness evaluation, we shall tune this value.

All the algorithms are implemented in Visual C++. Each experiment is run 5 times; the average is reported. We use a machine with an Intel Core 2.80 GHz CPU and 8GB RAM, running Windows 7.

9.2 Experimental Results

9.2.1 Time Efficiency of Inc-SR and Inc-uSR

We first evaluate the computational time of Inc-SR and Inc-uSR against Inc-SVD and Batch on real datasets.

Note that, to favor Inc-SVD that only works on small graphs (due to memory crash for high-dimension SVD $n > 10^5$), we just use Inc-SVD on DBLP and CitH.

Fig.7 depicts the results when edges are added to DBLP, CitH, YouTu, respectively. For each dataset, we fix $|V|$ and increase $|E|$ by $|\Delta E|$, as shown in the x-axis. Here, the edge updates are the differences between snapshots *w.r.t.* the "year" (*resp.* "video age") attribute of DBLP, CitH (*resp.* YouTu), reflecting their real-world evolution. We observe the following. (1) Inc-SR *always* outperforms Inc-SVD and Inc-uSR when edges are increased. For example, on DBLP, when the edge changes are 10.7%, the time for Inc-SR (83.7s) is 11.2x faster than Inc-SVD (937.4s), and 4.2x faster than Inc-uSR (348.7s). This is because Inc-SR deploys a rank-one matrix trick to update the similarities, with an effective pruning strategy to skip unnecessary recomputations, as opposed to Inc-SVD that entails rather expensive costs to incrementally update the SVD. The results on CitH are more pronounced, *e.g.,* Inc-SR is 30x better than Inc-SVD when $|E|$ is increased to 401K. (2) Inc-SR is consistently better than Batch when the edge changes are fewer than 19.7% on DBLP, and 7.2% on CitH. When link updates are 5.3% on DBLP (*resp.* 3.9%

[21] http://dblp.uni-trier.de/~ley/db/
[22] http://snap.stanford.edu/data/
[23] http://netsg.cs.sfu.ca/youtubedata/
[24] http://www.cse.ust.hk/graphgen/

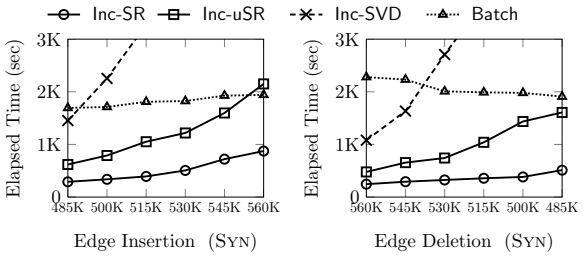


Fig. 9: Time Efficiency on Synthetic Data

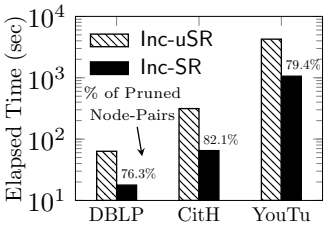


Fig. 10: Pruning Power

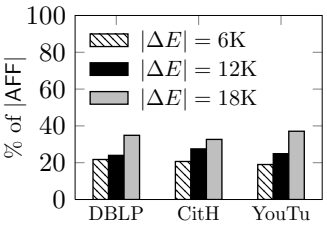


Fig. 11: % of Affected Areas

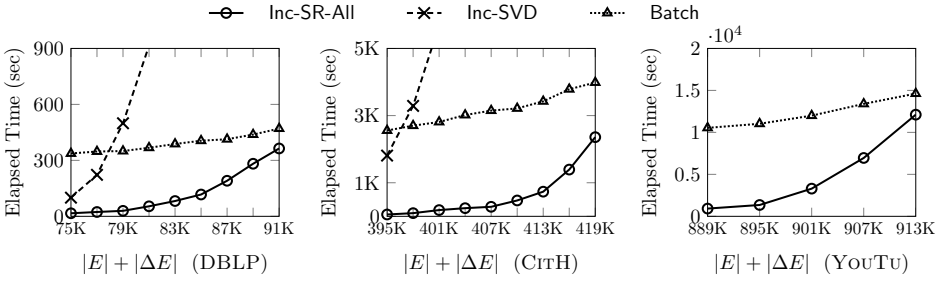


Fig. 12: Time Efficiency on Real Data (ΔE accompanies new nodes insertion)

Data ($\|E\|$)		Inc-bSR	Inc-SR-All	(%)
DBLP	75K	14.9	16.3	8.8
	83K	70.5	82.0	14.0
	91K	315.9	363.8	13.1
CitH	395K	50.5	54.5	7.3
	407K	241.9	283.5	14.6
	419K	1869.1	2357.4	20.7
YouTu	889K	876.6	921.9	4.9
	901K	2756.8	3297.4	16.4
	913K	10256.1	12109.2	15.3

Fig. 13: Time for Batch Updates

on CitH), Inc-SR improves Batch by 10.2x (*resp.* 4.9x). This is because (i) Inc-SR can exploit the sparse structure of $\mathbf{\Delta S}$ for incremental update, and (ii) small link perturbations may keep $\mathbf{\Delta S}$ sparsity. Hence, Inc-SR is highly efficient when link updates are small. (3) The computational time of Inc-SR, Inc-uSR, Inc-SVD, unlike Batch, is sensitive to the edge updates $|\Delta E|$. The reason is that Batch needs to reassess all similarities from scratch in response to link updates, whereas Inc-SR and Inc-uSR can reuse the old information in SimRank for incremental updates. In addition, Inc-SVD is too sensitive to $|\Delta E|$, as it entails expensive tensor products to compute SimRank from the updated SVD matrices.

Fig.8 shows the target rank r required for the *lossless* SVD of Eq.(5) *w.r.t.* the edge changes $|\Delta E|$ on DBLP and CitH. The y-axis is $\frac{r}{n} \times 100\%$, where $n = |V|$, and r is the rank of lossless SVD for $\mathbf{C}$ in Eq.(5). On each dataset, when increasing $|\Delta E|$ from 6K to 18K, we see that $\frac{r}{n}$ is 95% on DBLP (*resp.* 80% on CitH), Thus, r is not negligibly smaller than n in real graphs. Due to the quartic time *w.r.t.* r, Inc-SVD may be slow in practice to get a high accuracy.

On synthetic data, we fix $|V| = 79,483$ and vary $|E|$ from 485K to 560K (*resp.* 560K to 485K) in 15K increments (*resp.* decrements). The results are shown in Fig.9. We can see that, when 6.4% edges are increased, Inc-SR runs 8.4x faster than Inc-SVD, 4.7x faster than Batch, and 2.7x faster than Inc-uSR. When 8.8% edges are deleted, Inc-SR outperforms Inc-SVD by 10.4x, Batch by 5.5x, and Inc-uSR by 2.9x. This justifies our complexity analysis of Inc-SR and Inc-uSR.

9.2.2 Effectiveness of Pruning

Fig.10 shows the pruning power of Inc-SR as compared with Inc-uSR on DBLP, CitH, and YouTu, in which the percentage of the pruned node-pairs of each graph is depicted on the black bar. The y-axis is in a log scale. It can be discerned that, on every dataset, Inc-SR constantly outperforms Inc-uSR by nearly 0.5 order of magnitude. For instance, the running time of Inc-SR (64.9s) improves that of Inc-uSR (314.2s) by 4.8x on CitH, with approximately 82.1% node-pairs being pruned. That is, our pruning strategy is powerful to discard unnecessary node-pairs on graphs with different link distributions.

Since our pruning strategy hinges mainly on the size of the "affected areas" of the SimRank update matrix, Fig.11 illustrates the percentage of the "affected areas" of SimRank scores *w.r.t.* link updates $|\Delta E|$ on DBLP, CitH, and YouTu. We find the following. (1) When $|\Delta E|$ is varied from 6K to 18K on every real dataset, the "affected areas" of SimRank scores are fairly small. For instance, when $|\Delta E| = 12$K, the percentage of the "affected areas" is only 23.9% on DBLP, 27.5% on CitH, and 24.8% on YouTu, respectively. This highlights the effectiveness of our pruning method in real applications, where a larger number of elements of the SimRank update matrix with a-priori zero scores can be discarded. (2) For each dataset, the size of "affect areas" mildly grows when $|\Delta E|$ is increased. For example, on YouTu, the percentage of |AFF| increases from 19.0% to 24.8% when $|\Delta E|$ is changed from 6K to 12K. This agrees with our time efficiency analysis, where the speedup of Inc-SR is more obvious for smaller $|\Delta E|$.

Datasets	Inc-SR-All			Inc-bSR	Inc-SVD		
	No Optimization	Turn on Pruning	Turn on Column-wise Partitioning	Turn on Pruning & Column-wise Partitioning	$r = 5$	$r = 15$	$r = 25$
DBLP	722.5M	163.1M	1.3M	15.0M	1.36G	1.97G	3.86G
CitH	1.64G	413.9M	4.2M	34.8M	4.83G	—	—
YouTu	—	—	12.7M	186.2M	—	—	—

Fig. 14: Total Memory Efficiency on Real Data ("—" means memory explosion)

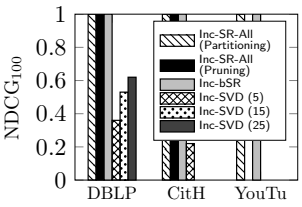


Fig. 15: Exactness

9.2.3 Time Efficiency of Inc-SR-All and Inc-bSR

We next compare the computational time of Inc-SR-All with Inc-SVD and Batch on DBLP, CitH, and YouTu. For each dataset, we increase $|E|$ by $|\Delta E|$ that might accompany new nodes insertion. Note that Inc-SR cannot deal with such incremental updates as $\mathbf{\Delta S}$ does not make any sense in such situations. To enable Inc-SVD to handle new nodes insertion, we view new inserted nodes as singleton nodes in the old graph G. Fig. 12 depicts the results. We can discern that (1) on every dataset, Inc-SR-All runs substantially faster than Inc-SVD and Batch when $|\Delta E|$ is small. For example, as $|\Delta E| = 6K$ on CitH, Inc-SR-All (186s) is 30.6x faster than Inc-SVD (5692s) and 15.1x faster than Batch (2809s). The reason is that Inc-SR-All can integrate the merits of Inc-SR with Inc-uSR-C1, Inc-uSR-C2, Inc-uSR-C3 to dynamically update SimRank scores in a rank-one style with no need to do costly matrix-matrix multiplications. Moreover, the complete framework of Inc-SR-All allows itself to support link updates that enables new nodes insertion. (2) When $|\Delta E|$ grows larger on each dataset, the time of Inc-SVD increases significantly faster than Inc-SR-All. This larger increase is due to the SVD tensor products used by Inc-SVD.In contrast, Inc-SR-All can effectively reuse the old SimRank scores to compute changes even if such changes may accompany new nodes insertion.

Fig. 13 compares the computational time of Inc-bSR with Inc-SR-All. From the results, we can notice that, on each graph, Inc-bSR is consistently faster than Inc-SR-All. The last column "(%)" denotes the percentage of Inc-bSR improvement on speedup. On each dataset, the speedup of Inc-bSR is more apparent when $|\Delta E|$ grows larger. For example, on DBLP, the improvement of Inc-bSR over Inc-SR-All is 8.8% when $|E| = 75$K, and 14.0% when $|E| = 83$K. This is because the large size of $|\Delta E|$ may increase the number of the new inserted edges with one endpoint overlapped. Hence, more such edges can be handled simultaneously by Inc-bSR, highlighting its high efficiency over the one-by-one update of Inc-SR-All.

9.2.4 Total Memory Usage

Fig. 14 evaluates the total memory usage of Inc-SR-All and Inc-bSR against Inc-SVD on real datasets. Note that the total memory usage includes the storage of the old SimRanks required for all-pairs dynamical evaluation. For Inc-SR-All, we test its three versions: (a) We first switch off our methods of "pruning" and "column-wise partitioning", denoted as "No Optimization"; (b) next turn on "pruning" only; and (c) finally turn on both. For Inc-SVD, we also tune the default target rank $r = 5$ larger to see how the memory space is affected by r.

The results tells us that (1) on each dataset when the memory of Inc-SVD and Inc-bSR does not explode, the total spaces of Inc-SR-All and Inc-bSR are consistently much smaller Inc-SVD whatever target rank r is. This is because, unlike Inc-SVD, Inc-SR-All and Inc-bSR need not memorize the results of SVD tensor products. (2) When the "pruning" switch is open, the space of Inc-SR-All can be reduced by $\sim$ 4x further on real data, due to our pruning method that discards many zeros in auxiliary vectors and final SimRanks. (3) When the "column-wise partitioning" switch is open, the space of Inc-SR-All can be saved by $\sim$ 100x further. The reason is that, as all pairs of SimRanks can be computed in a column-by-column style, there is no need to memorize the entire old SimRank $\mathbf{S}$ and auxiliary $\mathbf{M}$. This improvement agrees with our space analysis in Section 7. (4) The space of Inc-bSR is 8-11x larger than Inc-SR-All, but is still acceptable. This is because batch updates require more space to memoize several columns from the old $\mathbf{S}$ to handle a subset of edge updates simultaneously. (5) For Inc-SVD, when the target rank r is varied from 5 to 25, its total space increases from 1.36G to 3.86G on DBLP, but crashes on CitH and YouTu. This implies that r has a huge impact on the space of Inc-SVD, and is not negligible in the big-O analysis of [12].

9.2.5 Exactness

Finally, we evaluate the exactness of Inc-SR-All, Inc-bSR, and Inc-SVD on real datasets. We leverage the NDCG metrics [12] to assess the top-100 most similar pairs. We adopt the results of the batch algorithm [6] on each dataset as the NDCG_{100} baselines, due to its exactness. For Inc-SR-All, we evaluate its two enhanced versions: "with column-wise partitioning" and "with pruning"; for Inc-SVD, we tune its target rank r from 5 to 25.

Fig. 15 depicts the results, telling us the following. (1) On each dataset, the $NDCG_{100}$s of Inc-SR-All and Inc-bSR are 1, much better than Inc-SVD (< 0.62). This agrees with our observation in Section 3 that Inc-SVD may loss eigen-information in real graphs. In contrast, Inc-SR-All and Inc-bSR can guarantee the exactness. (2) The $NDCG_{100}$s for the two versions of Inc-SR-All are exactly the same, implying that both our pruning and column-wise partitioning methods are lossless while achieving high speedup.

10 Related Work

Recent results on SimRank computation can be distinguished into two broad categories: (i) incremental SimRank on dynamic graphs (*e.g.,* [8,12,21]), and (ii) batch SimRank on static graphs (*e.g.,* [5, 6, 10, 11, 13, 14, 20]).

10.1 Incremental SimRank

Generally, there are two types of dynamical algorithms: (i) deterministic method [8, 12], and (ii) probabilistic method [17]. Regarding deterministic approaches, the pioneering work of Li *et al.* [12] devised an excellent matrix representation of SimRank, and was the first to show a SVD method for incrementally updating all pairs of SimRanks, which requires $O(r^4n^2)$ time and $O(r^2n^2)$ memory, where r ($\leq n$) is the target rank of the low-rank approximation. However, (i) their incremental way is *inherently* inexact, with no guaranteed accuracy. It may miss some eigen-information (as we explained in Section 3) even though r is chosen to be exactly the full rank (instead of low rank) of the target matrix for the lossless SVD. (ii) In practice, r seems not much smaller than n for attaining a desired accuracy, but this may lead to prohibitively expensive updating costs for [12] because its time complexity $O(r^4n^2)$ is quartic *w.r.t.* r. In comparison, our work adopts a completely different framework from [12]. Instead of incrementally updating SVD, we first describe the changes to SimRank in response to every link update as a rank-one Sylvester equation, and then use graph topology to discard "unaffected areas" for speeding up the incremental computation of SimRank, without a compromise in accuracy. Our methods yield only linear time and memory *w.r.t.* n (independent of r) to incrementally compute all pairs of SimRanks for every link update. Moreover, for some types of link updates, *e.g.,* the insertion of edge (i, j) with i or j being a new node, the existing method by Li *et al.* [12] does not work effectively since their computational framework tacitly implies an assumption that new and old SimRank matrices must keep the same size. In contrast, our solution in this work can efficiently deal with such types of link updates, allowing new node insertions and deletions.

Another appealing piece of work is due to He *et al.* [8], who proposed the parallel computation of SimRank on digraphs, by leveraging an iterative aggregation method. Indeed, the parallel computing technique in [8] can be regarded as an efficient way to dynamically update new SimRank blocks. It differs from this work in that [8] is based on GPU to improve the parallel efficiency by reordering and splitting the first-order Markov chain, whereas our methods eliminate unnecessary recomputations in "unaffected areas" in terms of graph updates on CPU via a rank-one Sylvestor equation.

Most recently, Shao *et al.* [17] provided a picturesque exposition of a two-stage random walk sampling framework for SimRank search. In the preprocessing stage, they sampled a collection of one-way graphs to index random walks in a scalable manner. In the query stage, they retrieved similar nodes by pruning unqualified nodes based on the connectivity of one-way graph. The major advantage of their method is that one-way graph can be updated efficiently when the original graph changes. However, their methods deliver probabilistic results as they are based on a random walk sampling method.

There has also been a body of work on incremental computation of other graph-based relevance measures. Banhmani *et al.* [1] utilized the Monte Carlo method for incrementally computing Personalized PageRank. Desikan *et al.* [3] proposed an excellent incremental PageRank algorithm for node updating. Their central idea revolves around the first-order Markov chain. Sarma *et al.* [16] presented an excellent exposition of randomly sampling random walks of short length, and merging them together to estimate PageRank on graph streams. All these incremental methods are designed only for a specific measure, and may not be applied to SimRank.

10.2 Batch SimRank

In comparison to incremental algorithms, the batch SimRank computation has been well-studied on static graphs.

For deterministic methods, Jeh and Widom [9] were the first to propose an iterative paradigm for SimRank, entailing $O(Kd^2n^2)$ time for K iterations, where d is the average in-degree. Later, Lizorkin *et al.* [14] utilized the partial sums memoization to speed up SimRank computation to $O(Kdn^2)$. Recently, Yu *et al.* [20] have also improved SimRank computation to $O(Kd'n^2)$ time (with $d' \leq d$) via a fine-grained memoization to share the common parts among different partial sums. Fujiwara *et al.* [6] exploited the matrix form of [12] to

find the top-k similar nodes in $O(n)$ time *w.r.t.* a given query node. All these methods require $O(n^2)$ memory to output all pairs of SimRanks. Very recently, Kusumoto *et al.* [10] proposed an excellent linearized method that requires only $O(dn)$ memory and $O(K^2dn^2)$ time to compute all pairs of SimRanks. The recent work of [22] proposes an efficient aggregate method for computing partial pairs of SimRank scores. As a special case, the algorithm of [22] can evaluate all pairs of SimRanks in $O(Kdn^2)$ time and $O(Kdn)$ memory.

For probabilistic approaches, Fogaras and Rácz [5] proposed P-SimRank in linear time to estimate $s(a, b)$ from the probability that two random surfers, starting from a and b, will finally meet at a node. Li *et al.* [13] harnessed the random walks to compute local SimRank for a single node-pair. Later, Lee *et al.* [11] deployed the Monte Carlo method to find top-k SimRank node-pairs. Recently, Tao *et al.* [19] proposed an excellent two-stage way for the top-k SimRank-based similarity join.

11 Conclusions

In this article, we study the problem of incrementally updating SimRank scores on time-varying graphs. Our complete scheme, Inc-SR-All, consists of five ingredients: (1) For edge updates that do not accompany new nodes insertion, we characterize the SimRank update matrix $\mathbf{\Delta S}$ via a rank-one Sylvester equation. Based on this, a novel efficient algorithm is devised, which reduces the incremental computation of SimRank from $O(r^4n^2)$ to $O(Kn^2)$ for each link update. (2) To eliminate unnecessary SimRank updates further, we also devise an effective pruning strategy that can improve the incremental computation of SimRank to $O(K(nd + |\mathsf{AFF}|))$, where $|\mathsf{AFF}|$ ($\ll n^2$) is the size of "affected areas" in SimRank update matrix. (3) For edge updates that accompany new nodes insertion, we consider three insertion cases, according to which end of the inserted edge is a new node. For each case, we devise an efficient incremental SimRank algorithm that can support new nodes insertion and accurately update affected similarities. (4) To optimize the memory usage for all-pairs SimRank updates, we also devise a column-wise partitioning technique that significantly reduces the storage from $O(n^2)$ to $O(dn)$, without the need to memorize the entire old SimRank matrix. (5) For batch updates, we also propose efficient batch incremental methods that can handle "similar sink edges" simultaneously and eliminate redundant edge updates. The experimental evaluations on real and synthetic datasets demonstrate that (a) our incremental scheme is consistently 5-10x faster than Li *et al.*'s SVD based method; (b) our pruning strategy can speed up the incremental SimRank further by 3-6x; (c) the column-wise partitioning technique can reduce all pairs of SimRank computation by 100x memory usage; (d) the batch update algorithm enables an extra 5-15% speedup, with just a little compromise in memory.

References

1. B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized PageRank. *PVLDB*, 4(3), 2010.
2. P. Berkhin. Survey: A survey on PageRank computing. *Internet Mathematics*, 2, 2005.
3. P. K. Desikan, N. Pathak, J. Srivastava, and V. Kumar. Incremental PageRank computation on evolving graphs. In *WWW*, 2005.
4. D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, 2005.
5. D. Fogaras and B. Rácz. Practical algorithms and lower bounds for similarity search in massive graphs. *IEEE Trans. Knowl. Data Eng.*, 19, 2007.
6. Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for SimRank. In *ICDE*, 2013.
7. S. Garg, T. Gupta, N. Carlsson, and A. Mahanti. Evolution of an online social aggregation network: An empirical study. In *Internet Measurement Conference*, 2009.
8. G. He, H. Feng, C. Li, and H. Chen. Parallel SimRank computation on large graphs with iterative aggregation. In *KDD*, 2010.
9. G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *KDD*, 2002.
10. M. Kusumoto, T. Maehara, and K. Kawarabayashi. Scalable similarity search for SimRank. In *SIGMOD*, 2014.
11. P. Lee, L. V. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, 2012.
12. C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of SimRank for static and dynamic information networks. In *EDBT*, 2010.
13. P. Li, H. Liu, J. X. Yu, J. He, and X. Du. Fast single-pair SimRank computation. In *SDM*, 2010.
14. D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for SimRank computation. *PVLDB*, 1, 2008.
15. A. Ntoulas, J. Cho, and C. Olston. What's new on the web?: The evolution of the web from a search engine perspective. In *WWW*, 2004.
16. A. D. Sarma, S. Gollapudi, and R. Panigrahy. Estimating PageRank on graph streams. *J. ACM*, 58:13, 2011.
17. Y. Shao, B. Cui, L. Chen, M. Liu, and X. Xie. An efficient similarity search framework for SimRank over large dynamic graphs. *PVLDB*, 8(8):838–849, 2015.
18. Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. PathSim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4, 2011.
19. W. Tao, M. Yu, and G. Li. Efficient top-k SimRank-based similarity join. *PVLDB*, 8(3):317–328, 2014.
20. W. Yu, X. Lin, and W. Zhang. Towards efficient SimRank computation on large networks. In *ICDE*, 2013.
21. W. Yu, X. Lin, and W. Zhang. Fast incremental SimRank on link-evolving graphs. In *ICDE*, pages 304–315, 2014.
22. W. Yu and J. A. McCann. Efficient partial-pairs SimRank search for large networks. *PVLDB*, 8(5):569–580, 2015.
23. W. Yu and J. A. McCann. High quality graph-based similarity retrieval. In *SIGIR*, 2015.