View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Aston Publications Explorer

Fast Incremental SimRank on Link-Evolving Graphs

Weiren Yu [†,♭], Xuemin Lin [†], Wenjie Zhang [†]

[†] *The University of New South Wales, Sydney, Australia* [♭] *NICTA, Australia*
{weirenyu, lxue, zhangw}@cse.unsw.edu.au

***Abstract*—SimRank is an arresting measure of node-pair similarity based on hyperlinks. It iteratively follows the philosophy that 2 nodes are similar if they are referenced by similar nodes. Real graphs are often large, and links constantly evolve with small changes over time. This paper considers fast incremental computations of SimRank on link-evolving graphs. The prior approach [1] to this issue factorizes the graph via the singular value decomposition (SVD) first, and then incrementally maintains this factorization for link updates at the expense of exactness. Consequently, all node-pair similarities are estimated in $O(r^4n^2)$ time on a graph of n nodes, where r is the target rank of the low-rank approximation, which is not negligibly small in practice.**

In this paper, we propose a novel fast incremental paradigm. (1) We characterize the SimRank update matrix $\Delta\mathbf{S}$, in response to every link update, via a rank-one Sylvester matrix equation. By virtue of this, we devise a fast incremental algorithm computing similarities of n^2 node-pairs in $O(Kn^2)$ time for K iterations. (2) We also propose an effective pruning technique capturing the "affected areas" of $\Delta\mathbf{S}$ to skip unnecessary computations, without loss of exactness. This can further accelerate the incremental SimRank computation to $O(K(nd+|\mathsf{AFF}|))$ time, 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, $|\mathsf{AFF}| \ll n^2$. Our empirical evaluations verify that our algorithm (a) outperforms the best known link-update algorithm [1], and (b) runs much faster than its batch counterpart when link updates are small.

I. Introduction

With many recent eye-catching advances of the Internet, link analysis has become a common and important tool for web data management. Due to the proliferative applications (*e.g.,* link prediction, recommender systems, citation analysis), a surge of link-based similarity measures have come into play. For instance, Brin and Page [2] proposed a very successful relevance metric called Google PageRank to rank web pages. Jeh and Widom [3] introduced a novel measure, SimRank, to assess structural similarity between nodes based on hyperlinks. Sun *et al.* [4] invented PathSim to quantify node proximity for heterogeneous graphs. In these emerging similarity measures, SimRank has stood out as an arresting one over the last decade, due to its succinct and iterative philosophy that "two nodes are similar if they are referenced by similar nodes", coupled with the base case that "every node is maximally similar to itself". This recursion not only allows SimRank to capture the global structure of the graph, but also equips SimRank with appealing mathematical insight that has inspired research in recent years. For example, Fogaras and Rácz [5] interpreted SimRank as the first meeting time of the coalescing path-pair random walks. Li *et al.* [1] harnessed an elegant matrix equation to depict the closed form of SimRank.

Nevertheless, the batch computation of SimRank is costly: $O(Kd'n^2)$ time for all node-pairs [6], where K is the number of iterations, and $d' \leq d$ (d is the average graph in-degree). In general, real graphs are often large, with links constantly evolving with minor changes. This is particularly evident in *e.g.,* co-citation networks, web graphs, and social networks. As a statistical example [7], there are 5%–10% links updated every week in a web graph. It is rather expensive to reassess similarities for all pairs of nodes from scratch when the 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 skip unnecessary recomputations. Hence, we investigate the following problem in this paper.

Problem (Incremental SimRank Computation)
Given a graph G, the similarities $\mathbf{S}$ for G, the link changes ΔG [1] to G, and the damping factor $C \in (0,1)$.
Compute the changes $\Delta\mathbf{S}$ to the similarities $\mathbf{S}$.

In contrast with the work on batch SimRank computation, the study on incremental SimRank for link updates is limited. Indeed, due to the recursive nature of SimRank, it is hard to identify "affected areas" for incrementally updating SimRank. To the best of our knowledge, there is only one work [1] by Li *et al.* who gave an interesting method for finding the SimRank changes in response to link updates. Their idea is to factorize the backward transition matrix $\mathbf{Q}$ [2] of the original graph into $\mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T$ [3] via the 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. As a result, updating the similarities of all node-pairs entails $O(r^4n^2)$ time without guaranteed accuracy, where r $(\leq n)$ is the target rank of the low-rank approximation[4], which is not always negligibly small in practice, as illustrated in the following example.

Example 1. *Fig. 1 is a citation graph G (a fraction of* DBLP*) where each edge depicts a reference from one paper to another. Assume 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 want to compute SimRank scores in the new graph $G \cup \Delta G$. The existing method by Li et al. (see Algorithm 3 in [1]) first decomposes the old $\mathbf{Q} = \mathbf{U}\cdot\mathbf{\Sigma}\cdot\mathbf{V}^T$ as a precomputation step, then, when edge (i,j) is added, it incrementally updates the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$, and utilizes their updated versions to obtain*

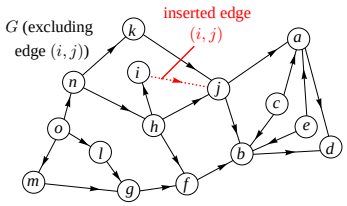
[1] ΔG consists of a sequence of edges to be inserted/deleted.

[2] In the notation of [1], the backward transition matrix $\mathbf{Q}$ is denoted as $\tilde{\mathbf{W}}$, which is the row-normalized transpose of the adjacency matrix.

[3] We use $\mathbf{X}^T$ (instead of $\tilde{\mathbf{X}}$ in [1]) to denote the transpose of matrix $\mathbf{X}$.

[4] According to [1], using our notations, $r \leq \text{rank}(\mathbf{\Sigma} + \mathbf{U}^T \cdot \Delta\mathbf{Q} \cdot \mathbf{V})$, where $\Delta\mathbf{Q}$ is the changes to $\mathbf{Q}$ for link updates.

[5] According to [3], the damping factor C is empirically set around 0.6–0.8, which indicates the rate of decay as similarity flows across edges.



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

Fig. 1: Compute SimRank incrementally as edge (i,j) is added

the new SimRank scores in $G \cup \Delta G$. The results are shown in Column '$\text{sim}_{\text{Li et al.}}$' of the table. For comparison, we also use a batch algorithm [6] to compute the "true" SimRank scores in $G \cup \Delta G$ from scratch, as illustrated in Column 'sim_{true}'. It can be noticed that for several node-pairs (not highlighted in gray), the similarities obtained by Li et al.'s incremental method [1] are different from the "true" SimRank scores even if the lossless SVD is used [6] *during the process of updating $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$, that is, Li et al.'s incremental approach [1] is inherently* approximate*. In fact, as will be rigorously explained in Section IV, their incremental strategy may miss some eigen-information whenever rank$(\mathbf{Q}) < n$.*

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

Example 1 tells that Li *et al.*'s incremental method [1] is approximate, and the $O(r^4n^2)$ time for updating all node-pair scores might be costly, as r is not always much smaller than n. Inspired by this, we propose a novel fast (exact[8]) algorithm for incrementally computing SimRank on link-evolving graphs. Instead of incrementally finding *the changes to the SVD of* $\mathbf{Q}$ for computing new similarities, our method can cope with the dynamic nature of link updates, by precomputing SimRank on the old entire graph once via a batch algorithm first, and then incrementally finding *SimRank updates* $\Delta\mathbf{S}$ *w.r.t.* link updates. Moreover, as links are often updated with small changes, not all node-pair similarities need to be updated. As an example in the table of Fig. 1, many node-pair similarities (highlighted in gray) remain unchanged when edge (i,j) is added. However, it is a grand challenge to identify the "affected areas" of $\Delta\mathbf{S}$, as SimRank is defined in a recursive fashion. To resolve this problem, we formulate $\Delta\mathbf{S}$ as an aggregation of similarities based on incoming paths. There are opportunities to find its "affected areas" by detecting the changes in these paths.

Contributions. Our main contributions are summarized below.

- We characterize the SimRank update matrix $\Delta\mathbf{S}$ *w.r.t.* every link update via a rank-one Sylvester matrix equation. In light of this, we devise a fast incremental algorithm that can update similarities of all n^2 node-pairs in $O(Kn^2)$ time for K iterations. (Section V-A)
- We also propose an effective pruning strategy to identify the "affected areas" of $\Delta\mathbf{S}$ to skip unnecessary similarity recomputations, without loss of exactness. This enables a further speedup in the incremental SimRank computation, which is in $O(K(nd + |\mathsf{AFF}|))$ time, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ $(\leq n^2)$ is the size of "affected areas", in practice, $|\mathsf{AFF}| \ll n^2$. (Section V-B)
- We conduct extensive experiments on real and synthetic datasets to demonstrate that our algorithm (a) consistently outperforms the best known link-incremental algorithm [1], from several times to over one order of magnitude, and (b) runs much faster than the batch counterpart [6] when link updates are small. (Section VI)

II. Related Work

SimRank is arguably one of the most appealing link-based similarity measures in a graph. Recent results on SimRank computation can be distinguished into two broad categories: (i) incremental SimRank on dynamic graphs (*e.g.,* [1], [8]), and (ii) batch SimRank on static graphs (*e.g.,* [6], [9]–[13]).

A. Incremental Algorithm

Incremental computation is useful as real graphs are often updated with small changes. However, few results are known about incremental SimRank computation, much less than their batch counterparts (*e.g.,* [5], [6], [9]–[15]). Generally, there are two types of updates for dynamic graphs: (i) link updates, and (ii) node updates. About link-incremental SimRank algorithms, we are merely aware of [1] by Li *et al.* who gave an excellent matrix representation of SimRank, and showed a SVD method for incrementally updating similarities of all node-pairs in $O(r^4n^2)$ time [9], where r $(\leq n)$ is the target rank of the low-rank approximation. However, (i) their incremental approach is *inherently* inexact, without guaranteed accuracy. It may miss some eigen-information (as proved in Section IV) even if r is chosen to be exactly the rank (instead of low-rank) of the target matrix for the lossless SVD. (ii) In practice, r is not *much* smaller than n for attaining the desirable accuracy. This may lead to prohibitively expensive updating costs for [1] since its time complexity $O(r^4n^2)$ is quartic *w.r.t.* r. In comparison, our work adopts a completely different framework from [1]. Instead of incrementally updating SVD [1], we characterize the changes to SimRank in response to each link update as a rank-one Sylvester equation first, and then exploit the link structure to prune "unaffected areas" for speeding up the incremental computation of SimRank, without loss of exactness, which only needs constant time (independent of r) to incrementally compute every node-pair similarity for each link update.

[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, which is given by the user.

If $\alpha = \text{rank}(\mathbf{X})$, then $\mathbf{X}_\alpha = \mathbf{X}$, and we call it the *lossless SVD* of $\mathbf{X}$.

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* of $\mathbf{X}$.

[7] As defined in [1], 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] Here, the "exactness" of our iterative algorithm means that it can converge to the exact SimRank solution as the number of iterations increases.

[9] According to the *proof* of Lemma 2 in [1], the time is actually $O(r^4n^2)$, though, the statement of Lemma 2 says "it is bounded by $O(n^2)$". Observing that $r \ll n$ is not often the case, we do not explicitly omit r^4 in $O(\star)$ here.

Another interesting piece of work is due to He *et al.* [8], who devised the parallel computation of SimRank on digraphs, by leveraging the iterative aggregation strategy. Indeed, the parallel computing technique in [8] can be regarded as an efficient *node*-incremental updating framework for SimRank. It differs from this work in that [8] improves the efficiency by reordering and parallelizing the first-order Markov chain for node updates on GPU, instead of capturing the "unaffected areas" of SimRank *w.r.t.* link updates, whereas our methods utilize pruning rules to eliminate unnecessary recomputations for links updates on CPU via a rank-one Sylvestor equation.

There has also been work on the incremental computations for other hyperlink-based relevance measures (*e.g.,* [16]–[19]). Desikan *et al.* [16] proposed an efficient incremental PageRank algorithm for node updating. Their underlying principle is based on the first-order markov model. Banhmani *et al.* [17] utilized the Monte Carlo method for incrementally computing Personalized PageRank. Sarma *et al.* [18] gave an excellent exposition of concatenating the short random walks for estimating PageRank with provable accuracy on graph streams. All of these incremental methods are *probabilistic* in nature, with the focus on node ranking, and hence cannot be directly applied in SimRank node-pair scoring. Fujiwara *et al.* [19] proposed K-dash for finding top-k highest Random Walk with Restart (RWR) proximity nodes for a given query, which involves a strategy to incrementally *estimate* proximity bounds. However, their incremental process is *approximate*.

B. Batch SimRank Computation

In contrast to the incremental algorithm, the batch SimRank computation has been well-studied on static graphs. Recent results on batch SimRank can be mainly categorized into (i) deterministic computation (*e.g.,* [1], [3], [6], [9], [13]), and (ii) probabilistic estimation (*e.g.,* [5], [10]–[12]). The deterministic methods may obtain similarities of high accuracy, but the time complexity is less desirable than the probabilistic approaches.

For deterministic methods, Jeh and Widom [3] are 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.* [13] utilized the partial sums memoization to speed up SimRank computation to $O(Kdn^2)$. Li *et al.* [1] proposed a novel non-iterative matrix formula for SimRank. Apart from the incremental SimRank requiring $O(r^4n^2)$ time, they also used a SVD method for computing batch SimRank in $O(\alpha^4n^2)$ time, where α is the target rank of matrix $\mathbf{Q}$. Most recently, Yu *et al.* [6] have further 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.* [9] exploited the matrix form of [1] to find the top-k similar nodes in $O(n)$ time.

For probabilistic approaches, Fogaras and Rácz [11] 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. In [5], the lower bounds are further analyzed for the accuracy of P-SimRank on large graphs. Li *et al.* [10] harnessed the random walks to compute local SimRank for a single node-pair. Lee *et al.* [12] deployed the Monte Carlo method to find top-k SimRank node-pairs.

III. Background of SimRank

In this section, we briefly revisit the SimRank background. Intuitively, the central theme of SimRank is that "two nodes are similar if their incoming neighbors are themselves similar". Based on this idea, there have emerged two forms of SimRank: iterative form (*e.g.,* [3], [13]), and matrix form (*e.g.,* [1], [8]).

(1) Iterative Form of SimRank. Given a digraph $G = (V, E)$ with a vertex set V and an edge set E, the SimRank similarity between two nodes a and b, denoted by $s(a,b)$, is defined as (i) $s(a,b) = 0$, if $\mathcal{I}(a) = \varnothing$ or $\mathcal{I}(b) = \varnothing$; (ii) otherwise,

$$s(a,b) = \begin{cases} 1, & a = b; \\ \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s(i,j), & a \neq b. \end{cases} \quad (1)$$

where $C \in (0,1)$ is a damping factor, $\mathcal{I}(a)$ is the in-neighbor set of node a, and $|\mathcal{I}(a)|$ is the cardinality of $\mathcal{I}(a)$.

The resulting sequence $\{s_k(a,b)\}_{k=0}^{\infty}$ converges to $s(a,b)$, the *exact* solution of Eq.(1).

(2) Matrix Form of SimRank. In matrix notations, SimRank can be formulated as follows.

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n, \quad (2)$$

where $\mathbf{S}$ is the similarity matrix whose entry $[\mathbf{S}]_{i,j}$ denotes the similarity score $s(i,j)$, $\mathbf{Q}$ is the backward transition matrix whose entry $[\mathbf{Q}]_{i,j} = 1/|\mathcal{I}(i)|$ if there is an edge from j to i, and 0 otherwise, and $\mathbf{I}_n$ is an $n \times n$ identity matrix.

The notation $(\star)^T$ denotes the matrix transpose.

The consistency of the two forms is discussed in [1].

In this paper, for ease of presentation, our incremental techniques are mainly based on the matrix form [1] of SimRank. The similar incremental paradigm can be ported to computing the iterative form [3] of SimRank.

IV. A Fly in the Ointment in [1]

In this section, we provide theoretical analysis to show that Li *et al.*'s incremental approach [1] is *approximate* in nature, which might miss some eigen-information even if the lossless SVD is utilized for computing SimRank.

The existing incremental method [1] computes SimRank by expressing similarity matrix $\mathbf{S}$ in terms of matrices $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$, where $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ are the decomposed matrices of $\mathbf{Q}$ via SVD:

$$\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T. \quad (3)$$

Then, when links are changed, [1] incrementally computes the new SimRank matrix $\tilde{\mathbf{S}}$ by updating the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ as

$$\tilde{\mathbf{U}} = \mathbf{U} \cdot \mathbf{U_C}, \quad \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma_C}, \quad \tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{V_C}, \text{[10]} \quad (4)$$

where $\mathbf{U_C}, \mathbf{\Sigma_C}, \mathbf{V_C}$ in Eq.(4) are the decomposed matrices of the auxiliary matrix $\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V}$ via SVD, *i.e.,*

$$\mathbf{C} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T, \quad (5)$$

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

However, in the above process, we observe that using Eq.(4) to update the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ may miss some eigen-information.

[10] In the sequel, we abuse a tilde to denote the updated version of a matrix, *e.g.,* $\tilde{\mathbf{U}}$ is the updated matrix of old $\mathbf{U}$ after link updates.

The main problem in [1] is that the derivation of Eq.(4) rests on the assumption that

$$\mathbf{U} \cdot \mathbf{U}^T = \mathbf{V} \cdot \mathbf{V}^T = \mathbf{I}_n. \quad (6)$$

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.

Example 2. *Consider the matrix* $\mathbf{Q} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$*, and its lossless SVD:* $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ *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*

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

whereas

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

[11] *Hence, when* $\mathbf{Q}$ *is not full-rank, Eq.*(6) *does not always hold, but one can prove that the following identity always holds:*

$$\mathbf{U}^T \cdot \mathbf{U} = \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I}_\alpha$$

since the SVD ensures that $\mathbf{U}$ *and* $\mathbf{V}$ *are* column-*orthonormal matrices, i.e., every two column-vectors, say* $\mathbf{x}_i$ *and* $\mathbf{x}_j$ *of* $\mathbf{U}$ *(resp.* $\mathbf{V}$*) satisfy* $\mathbf{x}_i{}^T \cdot \mathbf{x}_j = \begin{cases} 1, & i{=}j; \\ 0, & i{\neq}j. \end{cases}$ □

To clarify that Eq.(6) is involved in the derivation of Eq.(4), let us briefly recall from [1] the 4 steps for 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 $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q}$. Replacing $\mathbf{Q}$ by Eq.(3) yields

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

STEP 2. Premultiply by $\mathbf{U}^T$ and postmultiply by $\mathbf{V}$ on both sides of Eq.(7), and use the property $\mathbf{U}^T \cdot \mathbf{U} = \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I}_\alpha$.[12] It follows that

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

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

$$\mathbf{U}^T \cdot \tilde{\mathbf{Q}} \cdot \mathbf{V} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T. \quad (9)$$

STEP 4. Li *et al.* [1] 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

$$\underbrace{\mathbf{U} \cdot \mathbf{U}^T}_{\doteq\, \mathbf{I}_n} \cdot \tilde{\mathbf{Q}} \cdot \underbrace{\mathbf{V} \cdot \mathbf{V}^T}_{\doteq\, \mathbf{I}_n} = \underbrace{(\mathbf{U} \cdot \mathbf{U_C})}_{\triangleq \tilde{\mathbf{U}}} \cdot \underbrace{\mathbf{\Sigma_C}}_{\triangleq \tilde{\mathbf{\Sigma}}} \cdot \underbrace{(\mathbf{V_C}^T \cdot \mathbf{V}^T)}_{\triangleq \tilde{\mathbf{V}}^T}, \quad (10)$$

which is the result of Eq.(4).

However, the problem lies in STEP 4. As mentioned before, Eq.(6) does not hold when $\text{rank}(\mathbf{Q}) < n$. That is, for Eq.(10), $\tilde{\mathbf{Q}} \neq \tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T$. 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) in incrementally "approximating" $\tilde{\mathbf{S}}$.

Example 3. *Consider the old* $\mathbf{Q}$ *and its SVD in Example 2. Suppose there is an added edge, associated with* $\mathbf{\Delta Q} = \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$.

Li et al. [1] first computes the auxiliary matrix $\mathbf{C}$ *as*

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

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

$$\mathbf{C} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T \textit{ with } \mathbf{U_C} = \mathbf{\Sigma_C} = \mathbf{V_C} = [1].$$

Finally, Li et al. [1] update the new SVD of $\tilde{\mathbf{Q}}$ *via Eq.*(4) *as*

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

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*

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

This suggests that Li et al.'s incremental way [1] of updating $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ *is approximate (e.g.,* $\tilde{\mathbf{U}} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$*, as compared with its "true" version* $\hat{\mathbf{U}} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$*, misses the eigenvector* $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$*). Worse still, the approximation error is not small in practice as* $\|\tilde{\mathbf{Q}} - \tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T\|_2 = \|\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right] - \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]\|_2 = 1$. □

Our analysis tells that Eq.(6) holds only when (i) $\mathbf{Q}$ is full-rank, and (ii) the SVD of $\mathbf{Q}$ is lossless ($n = \text{rank}(\mathbf{Q}) = \alpha$). Only in this case, Li *et al.*'s incremental method [1] produces *exact* SimRank, which does not miss any eigen-information. However, the time complexity $O(r^4 n^2)$ of [1] would become $O(n^6)$, which is rather expensive. In practice, as evidenced by our statistical experiments on Stanford Large Network Dataset Collection (SNAP) [13], most real-life graphs are not full-rank, which is also in part demonstrated by our evaluations in Fig.2b. Thus, [1] produces the approximate solution in most cases.

V. OUR INCREMENTAL SOLUTION

We now propose our incremental techniques for computing SimRank, with the focus on handling *unit update* (*i.e.,* a single edge insertion or deletion). Since *batch update* (*i.e.,* a list of link insertions and deletions mixed together) can be decomposed into a sequence of unit updates, unit update plays a vital role in our incremental method.

The main idea of our solution is based on two methods.

(i) We first show that SimRank update matrix $\mathbf{\Delta S} \in \mathbb{R}^{n \times n}$ can be characterized as *a rank-one Sylvester matrix equation*[14]. By leveraging the rank-one structure of the matrix, we provide a novel efficient paradigm for incrementally computing $\mathbf{\Delta S}$, which only involves *matrix-vector* and *vector-vector* multiplications, as opposed to *matrix-matrix* multiplications to directly compute the new SimRank matrix $\tilde{\mathbf{S}}$.

(ii) In light of our representation of $\mathbf{\Delta S}$, we then identify the "affected areas" of $\mathbf{\Delta S}$ in response to link update $\mathbf{\Delta Q}$,

[11] The notation $\mathbf{I}_\alpha$ denotes the $\alpha \times \alpha$ identity matrix.

[12] As mentioned in Example 2, since $\mathbf{U} \in \mathbb{R}^{n \times \alpha}$ is *column*-orthonormal (not *row*-orthonormal), it follows that $\mathbf{U}^T \cdot \mathbf{U} = \mathbf{I}_\alpha$, whereas $\mathbf{U} \cdot \mathbf{U}^T \neq \mathbf{I}_n$.

[13] http://snap.stanford.edu/data/

[14] Given the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$, the Sylvester matrix equation in terms of $\mathbf{X} \in \mathbb{R}^{n \times n}$ takes the form: $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$. When $\mathbf{C}$ is a rank-α ($\leq n$) matrix, we call it *the rank-α Sylvester equation*.

and devise an effective pruning strategy to skip unnecessary similarity recomputations for link updates.

Before detailing our two methods in the subsections below, we introduce the following notations. (i) $\mathbf{e}_i$ denotes the $n \times 1$ unit vector with a 1 in the i-th entry and 0s in other entries. (ii) d_i denotes the in-degree of the node i in the old graph G. (iii) For a matrix $\mathbf{X}$, (a) $[\mathbf{X}]_{i,j}$ denotes the (i,j)-entry of $\mathbf{X}$, (b) $[\mathbf{X}]_{i,\star}$ the i-th row of $\mathbf{X}$, (c) $[\mathbf{X}]_{\star,j}$ the j-th column of $\mathbf{X}$.

A. Characterizing $\Delta\mathbf{S}$ via Rank-One Sylvester Equation

We first give the big picture, followed by rigorous proofs.

Main Idea. For every edge (i,j) update, we observe that $\Delta\mathbf{Q}$ is a *rank-one* matrix, *i.e.,* there exist two column vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{n\times 1}$ such that $\Delta\mathbf{Q} \in \mathbb{R}^{n\times n}$ can be decomposed into the *outer product*[15] of $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T.^{16} \tag{11}$$

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

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

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 column vectors: $\mathbf{u}$ is derived from Eq.(11), and $\mathbf{w}$ can be represented in terms of the old $\mathbf{Q}$ and $\mathbf{S}$ (we will provide their exact expressions later after some discussions); and $\tilde{\mathbf{Q}} = \mathbf{Q} + \Delta\mathbf{Q}$.

Thus, computing $\Delta\mathbf{S}$ 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 computing $\mathbf{M}$ via Eq.(13) involves only *matrix-vector* and *vector-vector* multiplications, which is a substantial improvement achieved by our observation that $(C \cdot \mathbf{u}\mathbf{w}^T) \in \mathbb{R}^{n\times n}$ in Eq.(13) is a *rank*-1 matrix, as opposed to the (full) *rank*-n matrix $(1-C) \cdot \mathbf{I}_n$ in Eq.(14). To further elaborate on this, we readily convert the recursive forms of Eqs.(13) and (14), respectively, into the following series forms: [17]

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

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

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

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

involving costly *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 *matrix-vector* and *vector-vector* operations $(\tilde{\mathbf{Q}}\mathbf{u}) \cdot (\tilde{\mathbf{Q}}\mathbf{w})^T$. More specifically, by leveraging two auxiliary vectors $\boldsymbol{\xi}_k, \boldsymbol{\eta}_k$, we adopt the following iterative paradigm to compute Eq.(15):

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

which only requires *matrix-vector* multiplications (*e.g.,* $\tilde{\mathbf{Q}} \cdot \boldsymbol{\xi}_k$) and *vector-vector* multiplications (*e.g.,* $\boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T$), without the need to perform *matrix-matrix* multiplications.

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 $\Delta\mathbf{S}$ via Eq.(13), rather than the *batch* computation of $\tilde{\mathbf{S}}$ via Eq.(14).

Finding $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for Eqs.(11) and (13). The challenging tasks in characterizing $\Delta\mathbf{S}$ for our incremental method are (i) to find the vectors $\mathbf{u}, \mathbf{v}$ in Eq.(11) for the *rank-one* decomposition of $\Delta\mathbf{Q}$, and (ii) to express the vector $\mathbf{w}$ in Eq.(13) in terms of the old matrices $\mathbf{Q}$ and $\mathbf{S}$ for guaranteeing that Eq.(13) is a *rank-one* Sylvester equation.

To find $\mathbf{u}$ and $\mathbf{v}$ in Eq.(11), we show the following theorem.

Theorem 1. *If there is an edge (i,j) inserted into G, then the change in $\mathbf{Q}$ is an $n \times n$ rank-one matrix, i.e., $\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$, where*

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

If there is an edge (i,j) deleted from G, then the change in $\mathbf{Q}$ can be decomposed as $\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$, where

$$\mathbf{u} = \begin{cases} \mathbf{e}_j & (d_j = 1) \\ \frac{1}{d_j-1}\mathbf{e}_j & (d_j > 1) \end{cases}, \quad \mathbf{v} = \begin{cases} -\mathbf{e}_i & (d_j = 1) \\ [\mathbf{Q}]_{j,\star}^T - \mathbf{e}_i & (d_j > 1) \end{cases} \tag{18}$$

Proof: Due to space limitations, we shall only prove the insertion case. A similar proof holds for the deletion case.

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

(ii) If $d_j > 0$, all the nonzero entries in $[\mathbf{Q}]_{j,\star}$ are $\frac{1}{d_j}$. Thus, for the inserted edge (i,j), the old $\mathbf{Q}$ can be converted into the new $\tilde{\mathbf{Q}}$ via 2 steps. (a) We change all nonzero entries of $[\mathbf{Q}]_{j,\star}$ from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$, by multiplying $\frac{d_j}{d_j+1}$ on the j-th row of $\mathbf{Q}$. Recall from *the elementary matrix property* that multiplying the j-th row of a matrix by $\alpha \neq 0$ can be accomplished by using $\mathbf{I} - (1-\alpha)\mathbf{e}_j\mathbf{e}_j^T$ as a left-hand multiplier on the matrix. Hence, after this step, $\mathbf{Q}$ is converted into the matrix $\mathbf{Q}'$, *i.e.,*

$$\mathbf{Q}' = (\mathbf{I} - (1 - \tfrac{d_j}{d_j+1})\mathbf{e}_j\mathbf{e}_j^T) \cdot \mathbf{Q} = \mathbf{Q} - \tfrac{1}{d_j+1}\mathbf{e}_j \cdot [\mathbf{Q}]_{j,\star}.$$

(b) We next update the (j,i)-entry of $\mathbf{Q}'$ from 0 to $\frac{1}{d_j+1}$,

[15] The *outer product* of the column 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.

[16] The explict expression of $\mathbf{u}$ and $\mathbf{v}$ will be given after a few discussions.

[17] 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 yields the new $\tilde{\mathbf{Q}}$, *i.e.,*

$$\tilde{\mathbf{Q}} = \mathbf{Q}' + \tfrac{1}{d_j+1}\mathbf{e}_j\mathbf{e}_i^T = \mathbf{Q} - \tfrac{1}{d_j+1}\mathbf{e}_j \cdot ([\mathbf{Q}]_{j,\star} - \mathbf{e}_i^T).$$

Since $\Delta\mathbf{Q} = \tilde{\mathbf{Q}} - \mathbf{Q}$, it follows that

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

which proves the case $d_j > 0$ in Eq.(17). ■

Example 4. *Consider the graph G in Fig. 1. Suppose there is an edge (i,j) inserted into G. As 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 in $\mathbf{Q}$ is a 15×15 rank-one matrix, which can be decomposed as $\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$ with

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

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

For every link update, Theorem 1 suggests that the change $\Delta\mathbf{Q}$ has a very special structure — the $n \times n$ *rank-one* matrix. More importantly, it finds a rank-one decomposition for $\Delta\mathbf{Q}$, by expressing the vectors $\mathbf{u}$ and $\mathbf{v}$ in terms of d_j and $[\mathbf{Q}]_{j,\star}^T$. It should be noted that such a rank-one decomposition is not unique, since for any scalar $\lambda \neq 0$, the vectors $\mathbf{u}' \triangleq \lambda \cdot \mathbf{u}$ and $\mathbf{v}' \triangleq \frac{\mathbf{v}}{\lambda}$ can be another rank-one decomposition for $\Delta\mathbf{Q}$. However, for any $\mathbf{u}$ and $\mathbf{v}$ that satisfy Eq.(11), there exists a vector $\mathbf{w}$ such that Eq.(13) is a *rank-one* Sylvester equation.

Capitalizing on Theorem 1, we are now ready to determine the expression of $\mathbf{w}$ in Eq.(13) in terms of the old $\mathbf{Q}$ and $\mathbf{S}$.

Theorem 2. *Suppose there is an edge (i,j) updated in G. Let $\mathbf{u}$ and $\mathbf{v}$ be the rank-one decomposition of $\Delta\mathbf{Q}$ in Theorem 1. Then, (i) there exists a vector $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ with*

$$\mathbf{y} = \mathbf{Q} \cdot \mathbf{z}, \quad \lambda = \mathbf{v}^T \cdot \mathbf{z}, \quad \mathbf{z} = \mathbf{S} \cdot \mathbf{v} \tag{19}$$

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

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

Proof: We show this by following the two steps:

(a) We find a recursion for the SimRank update matrix $\Delta\mathbf{S}$.

To characterize $\Delta\mathbf{S}$ in terms of the old $\mathbf{Q}$ and $\mathbf{S}$, we subtract Eq.(2) from Eq.(14), and apply $\Delta\mathbf{S} = \tilde{\mathbf{S}} - \mathbf{S}$, yielding

$$\Delta\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \tilde{\mathbf{Q}} \cdot \Delta\mathbf{S} \cdot \tilde{\mathbf{Q}}^T - C \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T. \tag{20}$$

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

$$\tilde{\mathbf{Q}} = \mathbf{Q} + \Delta\mathbf{Q} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T. \tag{21}$$

Then, we plug Eq.(21) into the term $C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T$ of Eq.(20), and simplify the result into

$$\Delta\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \Delta\mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{T} \tag{22}$$

$$\text{with } \mathbf{T} = \mathbf{u}(\mathbf{Q}\mathbf{S}\mathbf{v})^T + (\mathbf{Q}\mathbf{S}\mathbf{v})\mathbf{u}^T + (\mathbf{v}^T\mathbf{S}\mathbf{v})\mathbf{u}\mathbf{u}^T. \tag{23}$$

We can readily verify that matrix $\mathbf{T}$ is symmetric ($\mathbf{T} = \mathbf{T}^T$). 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, utilizing the auxiliary vectors $\mathbf{z}, \mathbf{y}$ and the scalar λ, Eq.(23) can be simplified into

$$\mathbf{T} = \mathbf{u} \cdot \mathbf{w}^T + \mathbf{w} \cdot \mathbf{u}^T, \quad \text{with } \mathbf{w} = \mathbf{y} + \tfrac{\lambda}{2}\mathbf{u}. \tag{24}$$

(b) We next convert the recursion of $\Delta\mathbf{S}$ into the series form.

One can readily verify that the solution $\mathbf{X}$ to the matrix equation $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$ has the following closed form:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C} \quad \Leftrightarrow \quad \mathbf{X} = \sum\nolimits_{k=0}^{\infty} \mathbf{A}^k \cdot \mathbf{C} \cdot \mathbf{B}^k \tag{25}$$

Thus, based on Eq.(25), the recursive definition of $\Delta\mathbf{S}$ in Eq.(22) naturally leads itself to the following 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.(24) yields

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

In light of Eq.(25), the series form of $\mathbf{M}$ in Eq.(15) satisfies the rank-one Sylvester recursive form of Eq.(13). ■

Theorem 2 obtains an exact expression for $\mathbf{w}$ in Eq.(13). To be precise, given $\mathbf{Q}$ and $\mathbf{S}$ in the old graph G, and an edge (i,j) updated 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 $\mathbf{u}, \mathbf{v}, \mathbf{Q}, \mathbf{S}$. Because of the existence of the vector $\mathbf{w}$, the Sylvester form of Eq.(13) being *rank-one* can be guaranteed. Henceforth, our aforementioned trick can be deployed to iteratively compute $\mathbf{M}$ in Eq.(15), needing no *matrix-matrix* multiplications.

Computing $\Delta\mathbf{S}$. Determining $\mathbf{w}$ via Theorem 2 is intended to speed up the incremental computation of $\Delta\mathbf{S}$. Indeed, for each link update, the *whole* process of computing $\Delta\mathbf{S}$ in Eq.(12), given $\mathbf{Q}$ and $\mathbf{S}$, needs no *matrix-matrix* multiplications at all. Specifically, the computation of $\Delta\mathbf{S}$ consists of two phases: (i) Given $\mathbf{Q}$ and $\mathbf{S}$, we compute $\mathbf{w}$ via Theorems 1 and 2. This phase merely includes the matrix-vector multiplications (*e.g.,* $\mathbf{Q}\mathbf{z}, \mathbf{S}\mathbf{v}$), 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 paradigm for Eq.(15), as mentioned earlier, can circumvent the *matrix-matrix* multiplications. Thus, taking (i) and (ii) together, it suffices to harness only *matrix-vector* and *vector-vector* operations in whole process of computing $\Delta\mathbf{S}$.

Leveraging Theorems 1 and 2, we are able to characterize the SimRank change $\Delta\mathbf{S}$, based on the following theorem.

Theorem 3. *When there is an edge (i,j) updated in G, then the SimRank change $\Delta\mathbf{S}$ can be characterized as*

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

$$\mathbf{M} = \sum\nolimits_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{e}_j \cdot \boldsymbol{\gamma}^T \cdot (\tilde{\mathbf{Q}}^T)^k, \tag{26}$$

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

(i) For the edge insertion, $\boldsymbol{\gamma} =$

$$\begin{cases} \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j & (d_j = 0) \\ \frac{1}{(d_j+1)}\Big(\mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \frac{1}{C} \cdot [\mathbf{S}]_{\star,j} + (\frac{\lambda}{2(d_j+1)} + \frac{1}{C} - 1) \cdot \mathbf{e}_j\Big) & (d_j > 0) \end{cases} \tag{27}$$

(ii) For the edge deletion, $\boldsymbol{\gamma} =$

$$\begin{cases} -\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}+\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j & (d_j=1)\\ \frac{1}{(d_j-1)}\left(\frac{1}{C}\cdot[\mathbf{S}]_{\star,j}-\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}+(\frac{\lambda}{2(d_j-1)}-\frac{1}{C}+1)\cdot\mathbf{e}_j\right) & (d_j>1)\end{cases} \tag{28}$$

and the scalar λ *can be derived from*

$$\lambda=[\mathbf{S}]_{i,i}+\tfrac{1}{C}\cdot[\mathbf{S}]_{j,j}-2\cdot[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}-\tfrac{1}{C}+1. \tag{29}$$

Proof: For space interests, we merely show insertion case.

(i) When $d_j=0$, by Eq.(17) in Theorem 1, $\mathbf{v}=\mathbf{e}_i,\ \mathbf{u}=\mathbf{e}_j$. Plugging them into Eq.(19) gets $\mathbf{z}=[\mathbf{S}]_{\star,i}$, $\mathbf{y}=\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}$, $\lambda=[\mathbf{S}]_{i,i}$. Thus, by virtue of $\mathbf{w}=\mathbf{y}+\frac{\lambda}{2}\mathbf{u}$ in Theorem 2, we have $\mathbf{w}=\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}+\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j$. Coupling this with Eq.(15), $\mathbf{u}=\mathbf{e}_j$, and Theorem 2 proves the case $d_j=0$ in Eq.(27).

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

$$\mathbf{v}=\mathbf{e}_i-[\mathbf{Q}]^T_{j,\star},\quad \mathbf{u}=\tfrac{1}{d_j+1}\cdot\mathbf{e}_j. \tag{30}$$

Substituting these back into Eq.(19) yields

$$\mathbf{z}=[\mathbf{S}]_{\star,i}-\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star},\quad \mathbf{y}=\mathbf{Q}\cdot[\mathbf{S}]_{\star,i}-\mathbf{Q}\cdot\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star},$$

$$\lambda=[\mathbf{S}]_{i,i}-2\cdot[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}+[\mathbf{Q}]_{j,\star}\cdot\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star}.$$

To simplify $\mathbf{Q}\cdot\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star}$ in $\mathbf{y}$, and $[\mathbf{Q}]_{j,\star}\cdot\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star}$ in λ, we postmultiply both sides of Eq.(2) by $\mathbf{e}_j$ to obtain

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

We also premultiply both sides of Eq.(31) by $\mathbf{e}_j^T$ to get

$$[\mathbf{Q}]_{j,\star}\cdot\mathbf{S}\cdot[\mathbf{Q}]^T_{j,\star}=\tfrac{1}{C}\cdot([\mathbf{S}]_{j,j}-1)+1. \tag{32}$$

Plugging Eqs.(31) and (32) into $\mathbf{y}$ and λ, respectively, and then plugging the resulting $\mathbf{y}$ and λ into $\mathbf{w}=\mathbf{y}+\frac{\lambda}{2}\mathbf{u}$ produce

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

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

Combining this with Eqs.(15), (30) shows the case $d_j>0$ for Eq.(27). Finally, taking (i) and (ii) together with Theorem 2 completes the proof for the link insertion case. ∎

For each link update, Theorem 3 provides a novel method to compute the incremental SimRank matrix $\Delta\mathbf{S}$, by utilizing the previous information of $\mathbf{Q}$ and $\mathbf{S}$ in the original graph G, as opposed to [1] that entails the incremental SVD maintenance. To *efficiently* compute $\Delta\mathbf{S}$ via Theorem 3, two tricks are worth mentioning. (i) We observe 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.(27) and (28) is actually the inner product $[\mathbf{Q}]_{j,\star}\cdot[\mathbf{S}]_{\star,i}$, being the term in Eq.(29). 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 at all, but involves the matrix-vector multiplications iteratively (*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: $\tilde{\mathbf{Q}}\cdot\boldsymbol{\eta}_k=\mathbf{Q}\cdot\boldsymbol{\eta}_k+(\mathbf{v}^T\cdot\boldsymbol{\eta}_k)\cdot\mathbf{u}$.

Algorithm. Based on Theorem 3, we provide an incremental SimRank algorithm, denoted as Inc-uSR, for each link update.

Given the old graph G, the old similarities $\mathbf{S}$ in G, the edge (i,j) updated to G, and the damping factor C, the algorithm

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

Input : a graph G, old similarities $\mathbf{S}$ for G, total #-iteration K, the edge (i,j) updated to G, and damping factor C.
Output: the new similarities $\tilde{\mathbf{S}}$ for $G\cup\{(i,j)\}$.

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** *edge* (i,j) *is to be inserted* **then**
6 **if** $d_j=0$ **then** $\mathbf{u}:=\mathbf{e}_j,\ \mathbf{v}:=\mathbf{e}_i,\ \boldsymbol{\gamma}:=\mathbf{w}+\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j$;
7 **else** $\mathbf{u}:=\frac{1}{d_j+1}\mathbf{e}_j,\quad \mathbf{v}:=\mathbf{e}_i-[\mathbf{Q}]^T_{j,\star}$;
8 $\boldsymbol{\gamma}:=\frac{1}{(d_j+1)}\big(\mathbf{w}-\frac{1}{C}[\mathbf{S}]_{\star,j}+(\frac{\lambda}{2(d_j+1)}+\frac{1}{C}-1)\mathbf{e}_j\big)$;
9 **else if** *edge* (i,j) *is to be deleted* **then**
10 **if** $d_j=1$ **then** $\mathbf{u}:=\mathbf{e}_j,\ \mathbf{v}:=-\mathbf{e}_i,\ \boldsymbol{\gamma}:=\frac{1}{2}[\mathbf{S}]_{i,i}\cdot\mathbf{e}_j-\mathbf{w}$;
11 **else** $\mathbf{u}:=\frac{1}{d_j-1}\mathbf{e}_j,\quad \mathbf{v}:=[\mathbf{Q}]^T_{j,\star}-\mathbf{e}_i$;
12 $\boldsymbol{\gamma}:=\frac{1}{(d_j-1)}\big(\frac{1}{C}[\mathbf{S}]_{\star,j}-\mathbf{w}+(\frac{\lambda}{2(d_j-1)}-\frac{1}{C}+1)\mathbf{e}_j\big)$;
13 initialize $\boldsymbol{\xi}_0:=C\cdot\mathbf{e}_j,\quad \boldsymbol{\eta}_0:=\boldsymbol{\gamma},\quad \mathbf{M}_0:=C\cdot\mathbf{e}_j\cdot\boldsymbol{\gamma}^T$;
14 **for** $k=0,1,\cdots,K-1$ **do**
15 $\boldsymbol{\xi}_{k+1}:=C\cdot\mathbf{Q}\cdot\boldsymbol{\xi}_k+C\cdot(\mathbf{v}^T\cdot\boldsymbol{\xi}_k)\cdot\mathbf{u}$;
16 $\boldsymbol{\eta}_{k+1}:=\mathbf{Q}\cdot\boldsymbol{\eta}_k+(\mathbf{v}^T\cdot\boldsymbol{\eta}_k)\cdot\mathbf{u}$;
17 $\mathbf{M}_{k+1}:=\boldsymbol{\xi}_{k+1}\cdot\boldsymbol{\eta}^T_{k+1}+\mathbf{M}_k$;
18 $\tilde{\mathbf{S}}:=\mathbf{S}+\mathbf{M}_K+\mathbf{M}_K^T$;
19 **return** $\tilde{\mathbf{S}}$;

incrementally computes the new similarities $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$. It works as follows. 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) vectors $\mathbf{u}$ and $\mathbf{v}$ for a rank-one factorization of $\Delta\mathbf{Q}$, and (ii) initial vector $\boldsymbol{\gamma}$ for subsequent $\mathbf{M}_k$ iterations (lines 5–12). Then, the algorithm maintains two auxiliary vectors $\boldsymbol{\xi}_k$ and $\boldsymbol{\eta}_k$ to iteratively compute matrix $\mathbf{M}_k$ (lines 13–17). The process continues until the number of iterations reaches a given K. Finally, the new scores $\tilde{\mathbf{S}}$ are obtained by $\mathbf{M}_K$[18] (line 18).

Example 5. *Recall the old graph* G *and* $\mathbf{S}$ *of* G *from Fig. 1. When edge* (i,j) *is added, we show how Inc-uSR computes the new* $\tilde{\mathbf{S}}$*, which is in part depicted in Column '*$\mathsf{sim}_{\mathsf{true}}$*'.*

Given the following information from the old $\mathbf{S}$ *below:* [19]

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

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

as $d_j=2$*, Inc-uSR first precomputes* $\mathbf{w}$ *and* λ *via lines 3–4:*

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

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

As an "edge insertion" operation, the vectors $\mathbf{u}$ *and* $\mathbf{v}$ *for a rank-one decomposition of* $\Delta\mathbf{Q}$ *can be computed via line 7. Their results are depicted in Example 4.*

[18] It can be proved that $\|\mathbf{M}_K-\mathbf{M}\|_{\max}\le C^{K+1}$, with $\mathbf{M}$ in Eq.(26).

[19] Due to space limitations, we only show the i-th and j-th columns of $\mathbf{S}$ here, which is sufficient for computing the new $\tilde{\mathbf{S}}$ in $G\cup\{(i,j)\}$.

Utilizing $\mathbf{w}$ *and* λ*, the vector* $\boldsymbol{\gamma}$ *can be obtained via line 8:*

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

Then, in light of $\boldsymbol{\gamma}$*,* Inc-uSR *iteratively computes* $\mathbf{M}_k$ *via lines 13–17. After* $K = 10$ *iterations,* $\mathbf{M}_K$ *is derived as*

	(a)	(b)	(c)	(d)	(e)	(f)	$\cdots$	(i)	(j)	$(k)\cdots(o)$
(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	
$\vdots$										
(i)		0					0		0	0
(j)	0.028	0.037	0	0		-0.068		-0.104	-0.060	
$\vdots$										
(o)		0					0		0	0

Finally, using $\mathbf{M}_K$ *and the old* $\mathbf{S}$*, the new* $\tilde{\mathbf{S}}$ *can be obtained via line 18, as partly shown in Column '*$\mathsf{sim}_{\mathsf{true}}$*' of Fig. 1.* □

Correctness & Complexity. (i) Algorithm Inc-uSR *correctly* updates the SimRank scores, which can be readily verified by Theorems 1–3. (ii) The total time of Inc-uSR can be bounded by $O(Kn^2)$ for updating *all* similarities of n^2 node-pairs.[20] To be specific, Inc-uSR runs in two phases: preprocessing (lines 1–12), and incremental iterations (lines 13–19). (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 only includes 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 15–16). Computing $\mathbf{M}_{k+1}$ entails $O(n^2)$ time for performing one outer product of two vectors and one matrix addition (lines 17). 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 [1] via SVD.

B. Pruning Unnecessary Node-Pairs in $\Delta\mathbf{S}$

After the SimRank update matrix $\Delta\mathbf{S}$ has been characterized in terms of a rank-one Sylvester equation, the pruning techniques in this subsection can further skip the node-pairs with unchanged similarities in $\Delta\mathbf{S}$ (*i.e.,* "unaffected areas"), avoiding unnecessary score recomputations for link update.

In practice, we observe that when link updates are small, affected areas in similarity updates $\Delta\mathbf{S}$ are often small as well. As demonstrated in Example 5, many entries in matrix $\mathbf{M}_K$ are 0s, implying that $\Delta\mathbf{S}$ $(= \mathbf{M}_K + \mathbf{M}_K^T)$ is a sparse matrix. However, it is a big challenge to identify such "affected areas" in $\Delta\mathbf{S}$ 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 the adjacency matrix of* G*. The entry* $[\mathbf{A}^k]_{i,j}$ *counts the number of length-*k *paths from node* i *to* j*.*

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

[20] In the next subsection, we shall further reduce the time complexity via a pruning strategy to eliminate node-pairs with unchanged similarities in $\Delta\mathbf{S}$.

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 weighted (*i.e.,* row-normalized) matrix of $\mathbf{A}^T$, we can verify $[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{i,j} = 0 \Leftrightarrow [(\mathbf{A}^T)^k\cdot\mathbf{A}^k]_{i,j} = 0$. 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 as follows:*

$$\underbrace{i \leftarrow \circ \leftarrow \cdots \leftarrow}_{\textit{length } k} \bullet \underbrace{\rightarrow \cdots \rightarrow \circ \rightarrow j}_{\textit{length } k}. \tag{33}$$

Definition 1. *We call the paths in Eq.*(33) the symmetric in-link paths of length $2k$ for node-pair (i, j).

By virtue of Eq.(25), the recursive form of SimRank Eq.(2) naturally leads itself to the following series form:

$$[\mathbf{S}]_{a,b} = (1-C)\cdot\sum\nolimits_{k=0}^{\infty} C^k\cdot[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{a,b}. \tag{34}$$

Capitalizing on Corollary 1, Eq.(34) 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.(34) 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 $[\mathbf{S}]_{i,j}$ into $[0, 1]$ since $\|\sum_{k=0}^{\infty} C^k\cdot[\mathbf{Q}^k\cdot(\mathbf{Q}^T)^k]_{a,b}\|_{\max} \le \sum_{k=0}^{\infty} C^k \le \frac{1}{1-C}$.

Affected Areas in $\Delta\mathbf{S}$. In light of our interpretation for $\mathbf{S}$ via Eq.(34), we next reinterpret the series $\mathbf{M}$ in Theorem 3, with the aim to identify the "affected areas" in $\Delta\mathbf{S}$.

Due to space limitations, we shall mainly focus on the edge insertion case of $d_j > 0$. Other cases have the similar results.

By substituting Eq.(27) (the case $d_j > 0$) back into Eq.(26), we can readily split the series form of $\mathbf{M}$ into three parts:

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

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) in graph $G\cup\{(i,j)\}$:

$$\overbrace{a \leftarrow \circ\cdots\circ \leftarrow j}^{[\tilde{\mathbf{Q}}^k]_{a,j}} \Leftarrow \overbrace{i \leftarrow \circ\cdots\circ \leftarrow \bullet \rightarrow \circ\cdots\circ \rightarrow \star}^{[\mathbf{S}]_{i,\star}} \overset{\mathbf{Q}^T}{\rightarrow} \overbrace{\blacktriangle \rightarrow \cdots \circ \rightarrow b}^{[(\tilde{\mathbf{Q}}^T)^k]_{\circ,b}} \tag{35}$$
$$\underbrace{\qquad\qquad}_{\text{length } k} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{\text{all symmetric in-link paths for node-pair } (i,\star)} \qquad \underbrace{\qquad\qquad}_{\text{length } k}$$

Such paths are the concatenation of four types of sub-paths (as depicted above) associated with four matrices, respectively, $[\tilde{\mathbf{Q}}^k]_{a,j}, [\mathbf{S}]_{i,\star}, \mathbf{Q}^T, [(\tilde{\mathbf{Q}}^T)^k]_{\circ,b}$, 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 new paths for node-pair (a,b):

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

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

$$\overbrace{a \leftarrow \circ \cdots \circ \leftarrow}^{[\tilde{\mathbf{Q}}^k]_{a,j}} j \overbrace{\rightarrow \circ \cdots \circ \rightarrow b}^{[(\tilde{\mathbf{Q}}^T)^k]_{j,b}} \quad (37)$$

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 $\Delta\mathbf{S}$). 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 $\mathbf{M}$.

To find the "affected areas" of $\mathbf{M}$, we prune the redundant node-pairs in G, based on the following theorem.

Theorem 4. *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 17 of Algorithm 1, and let*

$$\mathcal{F}_1 := \{b \mid b \in \mathcal{O}(y),\ \exists y,\ s.t.\ [\mathbf{S}]_{i,y} \neq 0\} \quad (38)$$

$$\mathcal{F}_2 := \begin{cases} \varnothing & (d_j = 0) \\ \{y \mid [\mathbf{S}]_{j,y} \neq 0\} & (d_j > 0) \end{cases} \quad (39)$$

$$\mathcal{A}_k \times \mathcal{B}_k := \begin{cases} \{j\} \times (\mathcal{F}_1 \cup \mathcal{F}_2 \cup \{j\}) & (k=0) \\ \{(a,b) |\, a \in \tilde{\mathcal{O}}(x),\ b \in \tilde{\mathcal{O}}(y),\ \exists x,\ \exists y,\ s.t.\ [\mathbf{M}_{k-1}]_{x,y} \neq 0\} & (k>0) \end{cases} \quad (40)$$

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.(39)*, 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 for $d_j > 0$, due to space limitations. The proofs of other cases are similar.

For $k = 0$, it follows from Eq.(26) that $[\mathbf{M}_0]_{a,b} = [\mathbf{e}_j]_a [\gamma]_b$. 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), $[\mathbf{e}_j]_a = 0$ since $a \neq j$. Thus, $[\mathbf{M}_0]_{a,b} = 0$. For case (ii), $[\mathbf{e}_j]_a = 1$ since $a = j$. Thus, $[\mathbf{M}_0]_{a,b} = [\gamma]_b$, where $[\gamma]_b$ 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).

In the sequel, our goal is to show the 3 terms are all 0s. (a) For $b \notin \mathcal{F}_1$, by definition in Eq.(38), $b \in \mathcal{O}(y)$ for $\forall y$, we have $[\mathbf{S}]_{i,y} = 0$. Due to symmetry, $b \in \mathcal{O}(y) \Leftrightarrow y \in \mathcal{I}(b)$ [21], which implies that $[\mathbf{S}]_{i,y} = 0$ for $\forall y \in \mathcal{I}(b)$. 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.(39) 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 17 of Algorithm 1 is the first k-th partial sum of $\mathbf{M}$ in Eq.(26). Thus, $\mathbf{M}_{k+1}$ can be derived from $\mathbf{M}_k$ as follows:

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

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

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

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 $C \cdot [\mathbf{e}_j]_a [\gamma]_b$ in the above equation is obviously 0. (ii) For $(a,b) \notin \mathcal{A}_k \times \mathcal{B}_k$, by virtue of Eq.(40), $a \in \tilde{\mathcal{O}}(x), b \in \tilde{\mathcal{O}}(y)$, 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 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 4 provides a pruning strategy to iteratively eliminate node-pairs with a-priori zero values in $\mathbf{M}_k$ (thus in $\Delta\mathbf{S}$). Hence, by leveraging Theorem 4, 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 $\Delta\mathbf{S}$.

Intuitively, $\mathcal{F}_1$ in Eq.(38) captures the nodes "▲" in (35). 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 (35). (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.(39), in the case of $d_j > 0$, consists of the nodes "⋆" depicted in (36). When $d_j = 0$, $\mathcal{F}_2 = \varnothing$ since the term $[\mathbf{S}]_{\star,i}$ does not appear in the expression of γ in Eq.(27) for the case when $d_j = 0$, in contrast with 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.(40). 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 4 hinges on $\Delta\mathbf{S}$ sparsity. Since real graphs are constantly updated with *minor* changes, $\Delta\mathbf{S}$ is often *sparse* in general. Hence, a huge body of node-pairs with zero scores in $\Delta\mathbf{S}$ can be eliminated in practice. As demonstrated by our experiments in Fig.2d, 76.3% paper-pairs on DBLP can be pruned, significantly reducing unnecessary similarity recomputations in response to link updates.

Example 6. *Recall Example 5 and the old graph G in Fig. 1. When edge (i,j) is inserted to G, according to Theorem 4, $\mathcal{F}_1 = \{a, b\}$, $\mathcal{F}_2 = \{f, i, j\}$, $\mathcal{A}_0 \times \mathcal{B}_0 = \{j\} \times \{a, b, f, i, j\}$. Hence, instead of computing the entire vector γ in Eq.*(27)*, we only need to compute part of its entries $[\gamma]_x$ for $\forall x \in \mathcal{B}_0$.*

[21] Recall that, as mentioned before, $\mathcal{I}(a)$ is the in-neighbor set of node a.

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.(40) ;
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 "$[\gamma]_b := [\mathbf{w}]_b + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot [\mathbf{e}_j]_b$, for all $b \in \mathcal{B}_0$" ;

13 set $[\boldsymbol{\xi}_0]_j := C,\ [\boldsymbol{\eta}_0]_b := [\gamma]_b,\ [\mathbf{M}_0]_{j,b} := C \cdot [\gamma]_b, \forall b \in \mathcal{B}_0$;

14 **for** $k = 1, \cdots, K$ **do**

15 find $\mathcal{A}_k \times \mathcal{B}_k$ via Eq.(40) ;

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** $\tilde{\mathbf{S}}$;

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. 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 $\Delta\mathbf{S}$, which is verified by Theorem 4. 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.(40), 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.2e, 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.

VI. Experimental Evaluation

We present an empirical study, using real and synthetic data, to show (i) the efficiency of Inc-SR for incremental computation in terms of time and space, as compared with (a) Inc-SVD, the best known link-update algorithm [1], (b) Inc-uSR, our incremental algorithm without pruning, and (c) Batch, the batch algorithm [6] via fine-grained memoization; (ii) the effectiveness of our pruning technique for identifying "affected areas" to speed up Inc-SR computation; and (iii) the exactness of Inc-SR and Inc-uSR, in contrast with Inc-SVD.

A. Experimental Settings

Datasets. We use both real and synthetic datasets.

(1) DBLP[22], a co-citation graph, where each node is a paper with attributes (*e.g.,* publication year), and edges are citations. By virtue of the year of the papers, we extract dense snapshots, each consisting of 93,560 edges and 13,634 nodes.

(2) CitH[23], a reference network (cit-HepPh) from e-Arxiv. If a paper u references v, the graph has one link from u to v. The dataset has 421,578 edges and 34,546 nodes.

(3) YouTu[24], a YouTube graph, where each node is a video. A video u is linked to v if v is in the related video list of u. We extract snapshots according to the age of the videos, and each has 953,534 edges and 178,470 nodes.

We use GraphGen[25] to build synthetic graphs and updates. The graphs are controlled by (a) the number of nodes $|V|$, and (b) the number of edges $|E|$. We produce the sequence of graphs following the linkage generation model [20]. Two parameters are utilized to control the updates: (a) update type (edge insertion/deletion), and (b) the size of updates $|\Delta G|$.

All the algorithms are implemented in Visual C++ v10.0. Each experiment is run 5 times; we report the average here. We use a machine with an Intel Core(TM) 2.80 GHz CPU and 8GB RAM, running Windows 7.

We set the decay factor $C = 0.6$, as in the prior work [3]. Our default iteration number is set to $K = 15$, with which a high accuracy $C^K \leq 0.0005$ is attainable, according to [13]; on large dataset YouTu, K is set to 5, the same value as [3]. For Inc-SVD, the target rank r is a time-accuracy trade-off; as shown in the experiments [1], the highest speedup is achieved when $r = 5$. Thus, in our time evaluations, $r = 5$ is adopted, whereas in the exactness evaluations, we shall tune this value.

B. Experimental Results

Exp-1: Time Efficiency. We first evaluate the running time of Inc-SR, Inc-uSR against Inc-SVD and Batch on real data.

To favor Inc-SVD that only works on graphs of small sizes (due to memory crash for high-dimension SVD, *e.g.,* $n > 10^5$), DBLP and CitH are used, though Inc-SR works well on a variety of graphs (*e.g.,* YouTu, Syn).

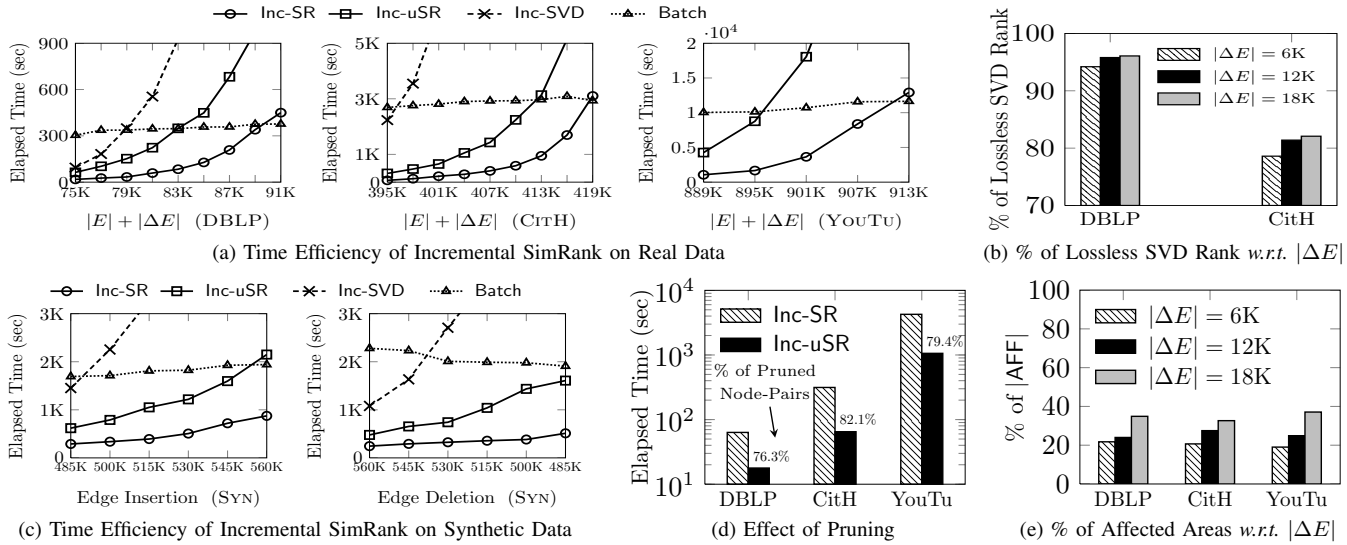
Fig.2a depicts the results for edges inserted into DBLP, CitH, YouTu, respectively. For each dataset, we fix $|V|$, and increase $|E|$ by $|\Delta E|$, as shown in the x-axis. Here, the edge updates are the differences between snapshots *w.r.t.* the "year" (*resp.* "video age") attribute of DBLP, CitH (*resp.* YouTu), reflecting their real-world evolution. We observe the following. (1) Inc-SR *always* outperforms Inc-SVD and Inc-uSR when

[22] http://dblp.uni-trier.de/~ley/db/

[23] http://snap.stanford.edu/data/

[24] http://netsg.cs.sfu.ca/youtubedata/

[25] http://www.cse.ust.hk/graphgen/



(a) Time Efficiency of Incremental SimRank on Real Data

(b) % of Lossless SVD Rank *w.r.t.* $|\Delta E|$

(c) Time Efficiency of Incremental SimRank on Synthetic Data

(d) Effect of Pruning

(e) % of Affected Areas *w.r.t.* $|\Delta E|$

Fig. 2: Performance Evaluations of Inc-uSR and Inc-SR on Real and Synthetic Datasets

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 about 30x better than Inc-SVD when $|E|$ is increased to 401K. On YouTu, Inc-SVD fails due to the memory crash for SVD. (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 the link updates are 5.3% on DBLP (*resp.* 3.9% on CitH), Inc-SR improves Batch by 10.2x (*resp.* 4.9x). This is because (i) Inc-SR exploits 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 running time of Inc-SR, Inc-uSR, Inc-SVD, unlike Batch, is sensitive to the edge updates $|\Delta E|$, as expected. 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 needs costly tensor products to compute SimRank from the updated SVD matrices.

Fig.2b 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 the 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 time being quartic *w.r.t.* r, Inc-SVD may be slow in practice to get a high accuracy.

Fixing $|V| = 79,483$ on synthetic data, we vary $|E|$ from 485K to 560K (*resp.* 560K to 485K) edges in 15K increments (*resp.* decrements). The results are reported in Fig.2c, confirming our observations on real datasets. For example, 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 the complexity analysis of our algorithms Inc-SR and Inc-uSR.

Exp-2: Effects of Pruning. As mentioned in Subsection V-B, Inc-SR skips needless computations for incremental updates.

To show the effectiveness of our pruning strategy in Inc-SR, we compare its time with that of Inc-uSR, *i.e.,* original version of Inc-SR without pruning rules, on DBLP, CitH, YouTu. The results are shown in Fig.2d, where the percentage of the pruned node-pairs in each graph is depicted on the black bar. The y-axis is in a logarithmic 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 (314.2s) improves that of Inc-uSR (64.9s) by 4.8x on CitH, with approximately 82.1% node-pairs being pruned. That is, our pruning technique is effective in finding unnecessary node-pairs on real graphs with various link distributions.

Since our pruning strategy hinges on the size of the "affected areas" in SimRank update matrix, it is imperative to evaluate, on real graphs, that how large these "affected areas" are when links are evolved. The results are visualized in Fig.2e, showing that the percentage of the "affected areas" in similarity changes *w.r.t.* link updates $|\Delta E|$ on real 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" in similarity changes are relatively small. For instance, when $|\Delta E| = 12K$, the percentage of the "affected areas" is only 23.9% on DBLP, 27.5% on CitH, and 24.8% on YouTu, respectively. This demonstrates the potential benefits of our pruning technique in real applications, where a larger number of elements in $\mathbf{\Delta S}$ with a-priori zero scores can be pruned. (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 confirms our observation in the time efficiency analysis, where Inc-SR speedup is more obvious for smaller $|\Delta E|$.

Exp-3: Memory Space. We next evaluate the memory requirements of Inc-SR, Inc-uSR, against Inc-SVD on real datasets.

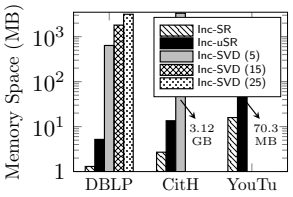


Fig. 3: Memory Space

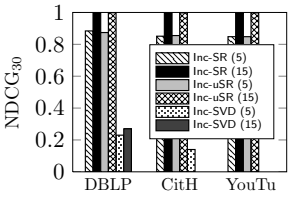


Fig. 4: NDCG$_{30}$ Exactness

Here, the memory space means "*intermediate* space", where the last step of writing n^2 node-pairs of the similarity outputs are not accommodated. We also tune the default target rank $r = 5$ larger for Inc-SR to see how memory increases *w.r.t.* r.

The results are depicted in Fig.3, where, for Inc-SR, we only report $r = 10$ and 15 on small DBLP, as its memory space will explode on larger networks when r and $|V|$ grow. We notice that (1) Inc-SR and Inc-uSR consume far smaller space than Inc-SR by at least 1.5 orders of magnitude on DBLP and CitH no matter what target rank r might be. This is because Inc-SR and Inc-uSR use the rank-one trick to convert $\mathbf{\Delta S}$ computations into the sequence of *vector* operations, whereas Inc-SVD needs to memoize the decomposed SVD matrices and to perform costly matrix tensor products. (2) Inc-SR has 4.1x (*resp.* 4.5x) smaller space than Inc-uSR on DBLP (*resp.* YouTu), due to our pruning method reducing the memoization of many entries in auxiliary vectors, *e.g.,* $\mathbf{w}$. (3) When r is varied from 5 to 25, the space of Inc-SVD is increased from 637.9M to 3.15G on DBLP, but crashes on CitH and YouTu. This tells that r has a large impact on the performance of Inc-SVD, which may not be ignored in the big-O notation of the complexity analysis [1]. Thus, to get Inc-SVD feasible on CitH, we use $r = 5$ in prior evaluations.

Exp-4: Exactness. Finally, we evaluate the exactness of Inc-SR and Inc-uSR against Inc-SVD. We adopt the NDCG metrics [1] to assess top-30 most similar node-pairs on DBLP, CitH, YouTu. For baselines of NDCG$_{30}$, we use the results of Batch on each dataset for 35 iterations.[26] For Inc-SR and Inc-uSR, we perform $K = 5, 15$ iterations on each graph; for Inc-SVD, due to its non-iterative paradigm, we tune the rank r from 5 to 15. The results are depicted in Fig.4, telling us the following. (1) In all the cases, Inc-SR and Inc-uSR have much better accuracy than Inc-SVD. For example, the NDCG$_{30}$ of Inc-SR and Inc-uSR are both 0.88 at $K = 5$, much better than Inc-SVD (0.36) at $r = 25$. This confirms our observations in Section IV, where we envision that Inc-SVD may miss some eigen-information in many real graphs. As $K = 10$, the NDCG$_{30}$ of Inc-SR and Inc-uSR are 1s, indicating that their top-30 node-pairs are perfectly accurate. This justifies the correctness of our algorithms. (2) For each dataset and the fixed iteration K, the NDCG$_{30}$ of Inc-SR and Inc-uSR are exactly the same. This indicates that our pruning strategy is lossless, *i.e.,* it does not sacrifice any exactness for speedup.

[26] As the diameters (*i.e.,* the longest paths) of DBLP, CitH, YouTu are 16,11,7, respectively, it suffices to perform $K = 35$ iterations to accommodate *all* path-pairs between two nodes for assessing SimRank. Thus, the resulting scores of Batch for $K = 30$ can be viewed as the *exact* baseline solutions.

VII. Conclusions

In this paper, we have proposed an efficient algorithm for incrementally computing SimRank on link-evolving graphs. Our algorithm, Inc-SR, is based on two ideas: (1) The SimRank update matrix $\mathbf{\Delta S}$ is characterized via a rank-one Sylvester equation. Based on this, a novel efficient paradigm is devised, which improves the incremental computation of SimRank from $O(r^4n^2)$ to $O(Kn^2)$ for every link update. (2) An effective pruning strategy is proposed to skip unnecessary similarity recomputations for link updates, which can further reduce the computation time of SimRank to $O(K(nd + |\mathsf{AFF}|))$, where $|\mathsf{AFF}|$ $(\leq n^2)$ is the size of "affected areas" in SimRank update matrix, which can be practically much smaller than n^2 in real evolution. Our empirical evaluations show that (1) Inc-SR consistently outperforms the best known link-update algorithm [1], from several times to over one order of magnitude, without loss of exactness. (2) Inc-SR runs substantially faster than its batch counterpart when link updates are small.

References

[1] 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.

[2] P. Berkhin, "Survey: A survey on PageRank computing," *Internet Mathematics*, vol. 2, 2005.

[3] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *KDD*, 2002.

[4] 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*, vol. 4, 2011.

[5] D. Fogaras and B. Rácz, "Practical algorithms and lower bounds for similarity search in massive graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 19, 2007.

[6] W. Yu, X. Lin, and W. Zhang, "Towards efficient SimRank computation on large networks," in *ICDE*, 2013.

[7] 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.

[8] G. He, H. Feng, C. Li, and H. Chen, "Parallel SimRank computation on large graphs with iterative aggregation," in *KDD*, 2010.

[9] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka, "Efficient search algorithm for SimRank," in *ICDE*, 2013.

[10] P. Li, H. Liu, J. X. Yu, J. He, and X. Du, "Fast single-pair SimRank computation," in *SDM*, 2010.

[11] D. Fogaras and B. Rácz, "Scaling link-based similarity search," in *WWW*, 2005.

[12] P. Lee, L. V. Lakshmanan, and J. X. Yu, "On top-k structural similarity search," in *ICDE*, 2012.

[13] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for SimRank computation," *PVLDB*, vol. 1, 2008.

[14] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei, "More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks," in *PVLDB*, 2014.

[15] I. Antonellis, H. G. Molina, and C. Chang, "SimRank++: query rewriting through link analysis of the click graph," *PVLDB*, vol. 1, 2008.

[16] P. K. Desikan, N. Pathak, J. Srivastava, and V. Kumar, "Incremental PageRank computation on evolving graphs," in *WWW*, 2005.

[17] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized PageRank," *PVLDB*, vol. 4, no. 3, 2010.

[18] A. D. Sarma, S. Gollapudi, and R. Panigrahy, "Estimating PageRank on graph streams," *J. ACM*, vol. 58, p. 13, 2011.

[19] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," *PVLDB*, vol. 5, 2012.

[20] S. Garg, T. Gupta, N. Carlsson, and A. Mahanti, "Evolution of an online social aggregation network: An empirical study," in *Internet Measurement Conference*, 2009.